Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
Fakulteta za matematiko in fiziko

Žiga Zorman

Primerjava algoritmov za izračun Fourierjeve transformacije s pomočjo sistema ALGator

DIPLOMSKO DELO

interdisciplinarni univerzitetni študijski program prve stopnje
računalništvo in matematika

Mentor: doc. dr. Tomaž Dobravec

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomskem delu se osredotočite na diskretno Fourierjevo transformacijo (DFT). Preglejte teoretična ozadja in podajte pregleden opis matematičnih področij, ki jih potrebujemo pri izračunu DTF. Preglejte in opišite tudi različne algoritme za izračun DFT in prikažite razliko med njimi (časovna/frekvenčna decimacija, različne osnove in podobno). Opisane algoritme implementirajte v jeziku Java in jih vključite v sistem ALGator ter opravite meritve kakovosti. Rezultate meritev prikažite in ovrednotite.

Izjava o avtorstvu zaključnega dela

Spodaj podpisani Žiga Zorman, z vpisno številko 63080411, sem avtor zaključnega dela z naslovom:

Primerjava algoritmov za izračun Fourierjeve transformacije s pomočjo sistema ALGator

IZJAVLJAM

1. da sem pisno zaključno delo študija izdelal samostojno pod mentorstvom doc. dr. Tomaža Dobravca;
2. da je tiskana oblika pisnega zaključnega dela študija istovetna elektronski obliki pisnega zaključnega dela študija;
3. da sem pridobil vsa potrebna dovoljenja za uporabo podatkov in avtorskih del v pisnem zaključnem delu študija in jih v pisnem zaključnem delu študija jasno označil;
4. da sem pri pripravi pisnega zaključnega dela študija ravnal v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobil soglasje etične komisije;
5. soglašam, da se elektronska oblika pisnega zaključnega dela študija uporabi za preverjanje podobnosti vsebine z drugimi deli s programsko opremo za preverjanje podobnosti vsebine, ki je povezana s študijskim informacijskim sistemom članice;
6. da na UL neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve avtorskega dela v elektronski obliki, pravico reproduciranja ter pravico dajanja pisnega zaključnega dela študija na voljo javnosti na svetovnem spletu preko Repozitorija UL;
7. dovoljujem objavo svojih osebnih podatkov, ki so navedeni v pisnem zaključnem delu študija in tej izjavi, skupaj z objavo pisnega zaključnega dela študija.

V Ljubljani, dne 20. februarja 2017 Podpis avtorja:

Za strokovno pomoč in nasvete pri izdelavi diplomskega dela se iskreno zahvaljujem mentorju doc. dr. Tomažu Dobravcu.
Posebna zahvala velja tudi staršem ter starim staršem, ki so mi stali ob strani in me spodbujali vse do zaključka študija.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
FT	Fourier transform	Fourierjeva transformacija
DFT	discrete Fourier transform	diskretna Fourierjeva transformacija
FFT	fast Fourier transform	hitra Fourierjeva transformacija
DIT	decimation in time	decimacija po času
DIF	decimation in frequency	decimacija po frekvenci

Povzetek

Naslov: Primerjava algoritmov za izračun Fourierjeve transformacije s pomočjo sistema ALGator

V diplomskem delu bomo najprej predstavili osnovne matematične pojme, ki so potrebni za razumevanje delovanja izbranih algoritmov. Potem pa se bomo poglobili v pet različnih implementacij Fourierjeve transformacije. Analizirali bomo njihovo časovno zahtevnost, število kompleksnih seštevanj in število kompleksnih množenj.
Zaradi integracije v sistem ALGator smo vse algoritme napisali v programskem jeziku Java. Sistem je namenjen razvijalcem algoritmov, saj omogoča učinkovito in enostavno testiranje ter analizo dobljenih rezultatov. Prav tako bomo med seboj primerjali implementacijo z decimacijo po času ter decimacijo po frekvenci enakega algoritma na rekurziven in iterativen način. Testni podatki, na katerih bomo testirali navedene algoritme, so sestavljeni iz vhodnih podatkov in dejanske rešitve. Za takšno obliko testov smo se odločili zato, da je preverjanje pravilnosti algoritmov čim hitrejše.

Ključne besede: Fourierjeva transformacija, kompleksna števila, N-ti primitivni koren enote, hitra Fourierjeva transformacija, diskretna Fourierjeva transformacija, Bluesteinov algoritem, osnova 2, osnova 4, deljena osnova, ALGator.

Abstract

Title: Comparison of Fourier transform algorithms with the help of ALGator system

At first we will explain the required mathematical knowledge which is crucial for the understanding of implemented algorithms in this thesis. Afterwards we will dive into five different implementations of Fourier transform and analyse their time complexity, number of complex additions and multiplications. Because of the integration in ALGator system all algorithms will be implemented in Java programming language. This system is intended to be used by the algorithm developers, because it offers very convenient implementation, effective testing and fast querying and analysis of the acquired results. We will compare decimation in time and decimation in frequency implementations of each algorithm and also its iterative and recursive forms. Testing data on which we will test our algorithms consist of the input data which needs to be transformed and correct transformation result. This way we will be able to verify the correctness of the algorithms as quickly as possible.

Keywords: Fourier transform, fast Fourier transform, discrete Fourier transform, complex numbers, N-th primitive root of unity, Bluestein's algorithm, radix-2, radix-4, split radix, ALGator.

Poglavje 1

Uvod

Zapleten signal lahko razdelimo na več preprostih valov. Fourierjeva transformacija (FT) nam pove vrsto in veličino valov iz katerih je sestavljen izvorni signal. V bistvu lahko vse na svetu predstavimo v obliki signala, na primer: zvočni signal, elektromagnetni signal, višino hriba v odvisnosti od lokacije, ceno neke dobrine v odvisnosti od časa itd. Fourierjeva transformacija nam omogoča edinstven vpogled in analizo teh signalov, zato je v uporabi na veliko znanstevenih področjih. Zaradi potrebe po čim hitrejši transformaciji vhodnega signala se je razvilo že veliko različnih algoritmov oziroma postopkov za izračun DFT.
FT ni omejena samo na funkcije časa, vendar bomo zaradi lažjega in razumljivejšega poimenovanja domeno vhodne funkcije poimenovali kot časovno domeno, domeno izhodne funkcije (FT) pa kot frekvenčno domeno.

1.1 Testiranje algoritmov

Poglavitna opravila tekom razvoja algoritma so meritve njegove učinkovitosti, primerjava z že obstoječimi algoritmi ter analiza izhodnih podatkov. Ravno zaradi naštetih nalog smo se odločili za uporabo sistema ALGator, ki vse to omogoča na enem mestu.

Prav tako pa so njegove pomembne lastnosti še:

- merjenje različnih časovnih parametrov (minimalni, maksimalni, povprečni, srednja vrednost),
- testiranje javanske kode,
- izvajanje algoritmov z različnimi vhodnimi podatki,
- shranjevanje rezultatov izvajanja testov v datotekah,
- grafični prikaz rezultatov izvajanja testov.

1.2 Implementirani algoritmi

V delu smo implementirali tri različne algoritme po metodi Cooley-Tukey, to so:

- algoritem z osnovo 2,
- algoritem z osnovo 4,
- algoritem z deljeno osnovo (2 in 4).

Za vse zgoraj naštete algoritme smo napisali dve različici, prva uporablja decimacijo v časvnem prostoru, druga pa decimacijo v frekvenčnem prostoru. Obe uporabljata rekurzivno metodo. Prav tako smo algoritma z osnovo 2 in osnovo 4 (z decimacijo v časovnem prostoru) napisali tudi v iterativni izvedbi. Poleg algoritmov, ki imajo omejitve glede dolžine vhodnih podatkov, smo implementirali še Bluesteinov algoritem ter za primerjavo še navadno DFT (po definiciji).
Cilj te naloge je med drugim tudi integracija problema računanja FT v sistem ALGator. To razvijalsko orodje se načeloma vsekozi izpopolnjuje in nadgrajuje, tako smo tudi mi v času nastajanja tega dela prispevali nekaj popravkov (napake v dokumentaciji, nepopoln regularni izraz za obravnavo poti do datotek na windows platformi, napaka z RSync orodjem) in nadgradenj (nova statistična funkcija).

1.3 Pregled dela

To diplomsko delo smo vključno z uvodom razdelili na šest vsebinsko zaokroženih poglavij.
Po uvodu v drugem poglavju navajamo teoretično predznanje (definicije in izreke), ki je potrebno za boljše razumevanje delovanja algoritmov. Natančneje smo opisali Eulerjevo formulo, N-ti primitivni koren enote ter matrike s posebnimi lastnostmi, ki jih uporablja Bluesteinov algoritem.
V tretjem poglavju se osredotočimo na natančnejšo predstavitev izbranih algoritmov, ki rešujejo dani problem ter navademo število kompleksnih seštevanj in množenj. Poglavje začnemo z definicijo DFT, potem opišemo algoritme, ki uporabljajo metodo Cooley-Tukey (z decimacijo v časovnem in tudi frekvenčnem prostoru), na koncu poglavja pa predstavimo idejo Bluesteinovega algoritma.
V četrtem poglavju predstavimo integracijo izbranih algoritmov v sistem ALGator. Začnemo s splošnimi informacijami o sistemu, nadaljujemo pa z natančnejšim opisom delovanja. Opisali smo naloge, ki jih mora administrator projekta v ALGatorju opraviti za pravilno integracijo algoritmov in problema v sistem. Razložili smo tudi strukturo testnih primerov ter vsebino pomožnih knjižnic in razredov, ki so potrebni za delovanje projekta znotraj ALGatorja.
Predzadnje poglavje opisuje in primerja rezultate testiranja, to so:

- čas izvajanja algoritmov,
- število kompleksnih seštevanj,
- število kompleksnih množenj,
- število rekurzivnih klicev (pri algoritmih implementiranih z rekurzijo),
- izračun časa izvajanja v odvisnosti od števila rekurzivnih klicev,
- primerjava časa računanja transformacije in inverzne transformacije.

V zadnjem poglavju predstavimo sklepne misli in navedemo ideje za nadaljne raziskovanja tega področja ter nekaj izboljšav sistema ALGator.

Poglavje 2

Definicije in izreki

2.1 Kompleksna števila

Učinkovito računanje DFT je zelo odvisno od razumevanja kompleksnih števil.
Kompleksna števila so bila predstavljena, da bi lahko rešili enačbe, kot so $x^2 + 1 = 0$, ki nimajo rešitve v množici realnih števil $\mathbb{R}$.

Definicija 2.1 *Kompleksno število je urejen par* (x, y) *kjer sta* $x, y \in \mathbb{R}$ *realni števili in na katerih sta operaciji seštevanje in množenje definirani kot [1]:*
Naj bosta (x_1, y_1) *in* (x_2, y_2) *kompleksni števili.*
Potem je kompleksno seštevanje $(x_1, y_1) + (x_2, y_2)$ *definirano kot:*

$$(x_1, y_1) + (x_2, y_2) := (x_1 + x_2, y_1 + y_2). \tag{2.1}$$

Kompleksno množenje $(x_1, y_1) \times (x_2, y_2)$ *pa kot:*

$$(x_1, y_1) \times (x_2, y_2) := (x_1x_2 - y_1y_2, x_1y_2 + y_1x_2). \tag{2.2}$$

V zgornji definiciji množenja dveh kompleksnih števil 2.2 opazimo, da potrebujemo za eno kompleksno množenje štiri realna množenja in dve realni seštevanji. Če označimo

$$A := x_1(x_2 + y_2), \tag{2.3}$$

$$B := y_2(x_1 + y_1), \tag{2.4}$$

$$C := x_2(y_1 - x_1), \tag{2.5}$$

potem lahko množenje dveh kompleksnih števil zapišemo s spodnjo enačbo

$$(x_1, y_1) \times (x_2, y_2) = (A - B, A + C). \tag{2.6}$$

Z načinom množenja dveh kompleksnih števil zapisanih v enačbi 2.6 potrebujemo za izračun tri realna množenja in pet realnih seštevanj. Opazimo, da smo prihranili eno realno množenje za ceno treh realnih seštevanj. Enačba 2.6 bi nam koristila samo v primeru, če bi se algoritmi izvajali na računalniku, ki izvede tri realna seštevanja hitreje kot eno realno množenje. V naših algoritmih bomo zato kompleksna množenja implementirali po enačbi 2.2.

Definicija 2.2 *Definirajmo imaginarno enoto i:*

$$i := (0, 1). \tag{2.7}$$

Iz enačbe (2.2) in definicije imaginarne enote 2.2 potem sledi:

$$i^2 = (0, 1) \times (0, 1) = (0 \cdot 0 - 1 \cdot 1, 0 \cdot 1 + 0 \cdot 1) = (-1, 0) = -1, \tag{2.8}$$

to nam pokaže, da je kvadratni koren imaginarne enote i enak -1. Opazimo tudi, da lahko kompleksno število (urejen par) zapišemo tudi kot vsoto realnega in imaginarnega dela:

$$(x, y) = (x, 0) + (y, 0) \times (0, 1) = (x, 0) + (0, y) = x + yi. \tag{2.9}$$

Izrek 2.1 *Naj bo $z = x + yi$ kompleksno število, kjer sta $x, y \in \mathbb{R}^2$. Potem absolutno vrednost števila z zapišemo kot $|z|$ in velja [1]:*

$$r = |z| = \sqrt{x^2 + y^2}. \tag{2.10}$$

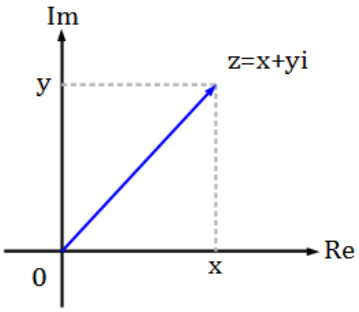
Ker je kompleksno število urejen par, ga lahko predstavimo v ravnini realnih števil $\mathbb{R}^2$.

2.1.1 Predstavitev kompleksnega števila v ravnini

Definicija 2.3 *Naj bo $z = x + yi$ kompleksno število, potem je:*

- *$Re(z) := x$ koeficient realnega dela kompleksnega števila z*

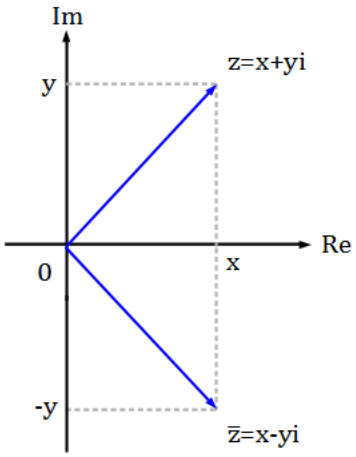
- *$Im(z) := y$ koeficient imaginarnega dela kompleksnega števila z*



Slika 2.1: Predstavitev kompleksnega števila v ravnini.

Definicija 2.4 *Konjugacija kompleksnega števila $z = x + yi$ je definirana kot [1]:*

$$\overline{z} = x - yi. \tag{2.11}$$



Slika 2.2: Predstavitev kompleksnega števila z in njegove konjugacije v ravnini.

2.1.2 Polarna predstavitev

Definicija 2.5 *Naj bo $z = x + yi$, kjer sta $x, y \in \mathbb{R}^2$. Potem kot, ki ga z oklepa z realno osjo, označimo [1]:*

$$\theta = arg(z) \tag{2.12}$$

Iz zgornje definicije 2.5, izreka o absolutni vrednosti kompleksnega števila 2.1 in trigonometričnih funkcij potem sledi:

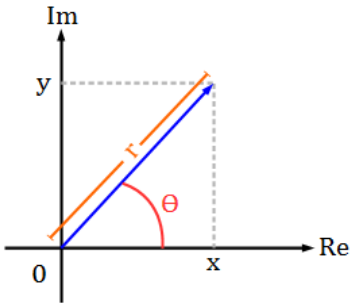
$$\frac{x}{r} = \cos\theta \implies x = r\cos\theta, \tag{2.13}$$

$$\frac{y}{r} = \sin\theta \implies y = r\sin\theta. \tag{2.14}$$

To pomeni, da lahko vsako kompleksno število $z = x + yi \neq 0$ zapišemo kot:

$$z = x + yi = r(\cos\theta + i\sin\theta). \tag{2.15}$$

Par $\langle r, \theta\rangle$ imenujemo polarna oblika kompleksnega števila $z \neq 0$. Število $z = 0 + 0i$ pa je definirano kot $\langle 0, 0\rangle$.



Slika 2.3: Predstavitev kompleksnega števila v polarnih koordinatah.

2.2 Eulerjeva formula

Za razumevanje Eulerjeve formule je potrebno omeniti Taylorjevo vrsto.

Izrek 2.2 *Naj bo funkcija f na okolici točke a (n+1)-krat zvezno odvedljiva, potem lahko funckijo f razvijemo v vrsto okrog točke a in velja [2]:*

$$f(x) = f(a)+f'(a)(x-a)+\frac{f''(a)}{2!}(x-a)^2+\ldots+\frac{f^{(n)}(a)}{n!}(x-a)^n+R_n(x) \quad (2.16)$$

$$R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-a)^{n+1}; \quad (2.17)$$

pri čemer je ξ neka točka med a in x.

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k + R_n(x) \quad (2.18)$$

Izrek 2.3 *Eulerjeva formula, včasih imenovana tudi Eulerjeva identiteta, se glasi [1]:*

$$e^{ix} = \cos(x) + i\sin(x) \quad (2.19)$$

kjer je i imaginarna enota, definirana z definicijo 2.2.

Dokaz. Za dokaz Eulerjeve formule si bomo pomagali s Taylorjevo vrsto. Najprej razvijmo Taylorjeve vrste za funckije $e^x, \cos(x)$ in $\sin(x)$ okrog točke $a = 0$, tako dobimo:

$$e^x = 1 + x + \frac{x^2}{2!} + \cdots = \sum_{k=0}^{\infty} \frac{x^k}{k!}, \quad (2.20)$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k}}{(2k)!}, \quad (2.21)$$

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)!}x^{2k+1}. \quad (2.22)$$

Sedaj, ko poznamo Taylorjevo vrsto za funkcijo e^x, lahko v funkcjo namesto x vstavimo ix, potem dobimo:

$$e^{ix} = 1 + ix + \frac{(ix)^2}{2!} + \frac{(ix)^3}{3!} + \frac{(ix)^4}{4!} + \frac{(ix)^5}{5!} + \ldots \quad (2.23)$$

Če se spomnimo enačbe (2.8) iz poglavja o kompleksnih številih, kjer smo pokazali, da velja $i^2 = -1$, potem lahko zgornjo enačbo poenostavimo:

$$e^{ix} = 1 + ix - \frac{x^2}{2!} - \frac{ix^3}{3!} + \frac{x^4}{4!} + \frac{ix^5}{5!} - \dots \tag{2.24}$$

Tako lahko izpostavimo i in dobimo:

$$e^{ix} = (1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \dots) + i(x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots). \tag{2.25}$$

Zaradi poznavanja Taylorjeve vrste funkcij $\cos(x)$ in $\sin(x)$ takoj opazimo, da vsota v prvem oklepaju predstavlja vrsto za $\cos(x)$, vsota v drugem oklepaju pa vrsto za $\sin(x)$ in tako potem dobimo Eulerjevo formulo:

$$e^{ix} = \cos(x) + i\sin(x). \tag{2.26}$$

□

2.3 N-ti primitivni koren enote

Definicija 2.6 *Naj bo K obseg in naj za $\omega \in K$ velja:*

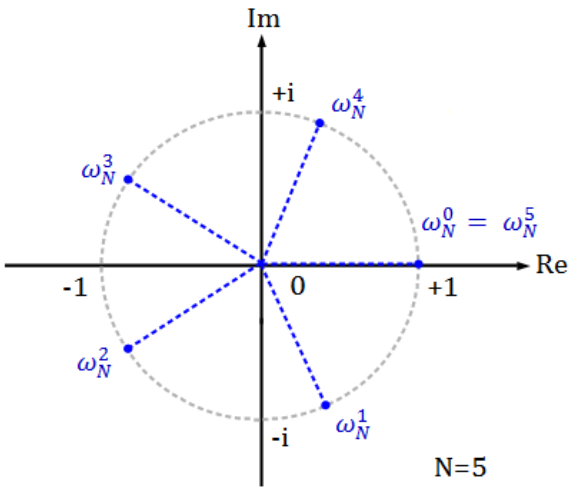
(i) $\omega^N = 1$,

(ii) $\omega^j \neq 1$; $j = 1, 2, \ldots, N-1$.

Potem se ω imenuje N-ti primitivni koren enote in ga označimo z ω_N [1].

V našem primeru kjer je obseg K = $\mathbb{C}$ (množici kompleksnih števil), bo primitivni N-ti koren enote [7]:

$$\omega_N = e^{-i\frac{2\pi}{N}} \tag{2.27}$$



Slika 2.4: Prikaz N-tega primitivnega korena enote v primeru N = 5.

Sedaj bomo navedli lastnosti N-tega primitivnega korena enote v množici $\mathbb{C}$, ki nam bodo omogočale učinkovitejši izračun FT [8].

(a) Periodičnost: $\omega_N^{k+N} = \omega_N^k$.

(b) Zrcalnost: $\omega_N^{k+\frac{N}{2}} = -\omega_N^k$.

(c) Simetrija kompleksne konjugiranosti: $\omega_N^{k(N-n)} = \overline{\omega_N^{kn}}$.

(d) $\omega_N^k = \omega_{\frac{N}{k}}$

(e) Če velja: $N \mod 2 = 0 \implies \omega_N^{\frac{N}{2}} = -1$.

V enakem vrstnem redu, kot smo našteli zgornje lastnosti, jih bomo sedaj dokazali.

Dokazi.

(a) Periodičnost dokažemo enostavno, če razširimo zapis na levi strani enačbe (a) s pomočjo Eulerjeve formule 2.3:

$$\begin{aligned} \omega_N^{k+N} &= \omega_N^k \cdot \omega_N^N = \omega_N^k \cdot e^{-i\frac{2\pi N}{N}} \\ &= \omega_N^k \cdot (\cos(-2\pi) + i\sin(-2\pi)) \\ &= \omega_N^k \cdot (1+0) \\ &= \omega_N^k. \end{aligned} \tag{2.28}$$

□

(b) Enako naredimo tudi pri dokazu zrcalnosti, razširimo levo stran enačbe (b):

$$\begin{aligned} \omega_N^{k+\frac{N}{2}} &= \omega_N^k \cdot \omega_N^{\frac{N}{2}} \\ &= \omega_N^k \cdot e^{-i\frac{2\pi}{N}\frac{N}{2}} \\ &= \omega_N^k \cdot e^{-i\pi} = \omega_N^k \cdot (\cos(-\pi) + i\sin(-\pi)) \\ &= \omega_N^k \cdot (-1+0) \\ &= -\omega_N^k. \end{aligned} \tag{2.29}$$

□

(c) Z upoštevanjem Eulerjeve formule 2.3 in definicije 2.4 dokažemo lastnost (c):

$$\begin{aligned}
\omega_N^{k(N-n)} &= \omega_N^{kN-kn} = e^{-i\frac{2\pi}{N}kN} \cdot \omega_N^{-kn} \\
&= \overline{(\cos(-2\pi k) + i\sin(-2\pi k))} \cdot e^{i\frac{2\pi}{N}kn} = (1+0)\cdot e^{i\frac{2\pi}{N}kn} \\
&= \overline{(\cos(\frac{2\pi}{N}kn) - i\sin(\frac{2\pi}{N}kn))} \\
&= \overline{e^{-i\frac{2\pi}{N}kn}} \\
&= \overline{\omega_N^{kn}}. \qquad (2.30)
\end{aligned}$$

□

(d) Z upoštevanjem enačbe (2.27) in Eulerjeve formule 2.3 dokažimo lastnost (d):

$$\begin{aligned}
\omega_N^k &= e^{-i\frac{2\pi}{N}k} \\
&= e^{-i\frac{2\pi}{N}\frac{1}{k}} = e^{-i\frac{2\pi}{\frac{N}{k}}} \\
&= \omega_{\frac{N}{k}}. \qquad (2.31)
\end{aligned}$$

□

(e) Dokažimo lastnost (e). Ker $N \mod 2 = 0 \implies N = 2k \; za \; \forall k \in \mathbb{N}$, potem lahko ω_N zapišemo kot:

$$\omega_N = e^{-i\frac{2\pi}{N}} = e^{-i\frac{2\pi}{2k}} = e^{-i\frac{\pi}{k}}. \qquad (2.32)$$

Zato velja:

$$\omega_N^{\frac{N}{2}} = \omega_N^k = e^{-i\frac{\pi}{k}k} = e^{-i\pi} = -1. \qquad (2.33)$$

□

2.4 Vrste matrik

Definicija 2.7 *Naj bo D kvadratna matrika velikosti $n \times n$. Matriko D imenujemo diagonalna matrika, če za vsak $i, j = 1, 2, \ldots, n$ velja $d_{i,j} = 0$, kjer $i \neq j$.*

$$D = diag(d_{1,1}, d_{2,2}, \ldots, d_{n,n}) = \begin{bmatrix} d_{1,1} & 0 & \ldots & 0 \\ 0 & d_{2,2} & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & d_{n,n} \end{bmatrix}. \tag{2.34}$$

Definicija 2.8 *Za kvadratno matriko A velikosti $n \times n$ rečemo, da je simetrična, če je enaka svoji transponirani obliki. Elementi simetrične matrike so enaki glede na glavno diagonalo [3]. Primer simetrične matrike velikosti 4×4:*

$$A = A^T = \begin{bmatrix} a & b & c & d \\ b & e & f & g \\ c & f & h & i \\ d & g & i & j \end{bmatrix}. \tag{2.35}$$

Definicija 2.9 *Toeplitzova matrika je kvadratna matrika velikosti $n \times n$, $T_n = [t_{i,j};\ i, j = 0, 1, \ldots, n-1]$, kjer velja $t_{i,j} = h_{i-j}$ za vseh $2n-1$ skalarjev $h_{-(n-1)}, \ldots, h_0, \ldots, h_{n-1}$. kot na primer [4]:*

$$T_n = \begin{bmatrix} h_0 & h_{-1} & h_{-2} & \ldots & h_{-(n-1)} \\ h_1 & h_0 & h_{-1} & & \\ h_2 & h_1 & h_0 & & \vdots \\ \vdots & & & \ddots & \\ h_{n-1} & & & \ldots & h_0 \end{bmatrix}. \tag{2.36}$$

Definicija 2.10 *Krožna matrika C je Toeplitzova matrika posebne oblike,*

$$C = \begin{bmatrix} c_0 & c_1 & c_2 & & \dots & c_{n-1} \\ c_{n-1} & c_0 & c_1 & c_2 & & \vdots \\ & c_{n-1} & c_0 & c_1 & \ddots & \\ \vdots & \ddots & \ddots & \ddots & & c_2 \\ & & & & & c_1 \\ c_1 & \dots & & c_{n-1} & & c_0 \end{bmatrix}, \tag{2.37}$$

kjer je vsaka vrstica ciklični zamik vrstice nad njo. Obliko matrike lahko opišemo tudi na način, če je (i, j) element matrike C, $C_{i,j}$, podan s spodnjo enačbo [4]:

$$C_{i,j} = c_{(j-i) \bmod n}. \tag{2.38}$$

Poglavje 3

Algoritmi

V tem poglavju si bomo podrobneje ogledali algoritme za izračun FT in število kompleksnih seštevanj in množenj, ki jih potrebujejo. Teoretične predstavitve si bodo sledile v vrstnem redu: algoritem z osnovo 2, algoritem z osnovo 4 ter algoritem z deljeno osnovo (2 in 4). Pri vsakem od naštetih algoritmov bomo med seboj primerjali DIT in DIF izvedbo. Na koncu poglavja pa bomo predstavili tudi glavno idejo Bluesteinovega algoritma.
Pri implementaciji algoritmov smo uporabili paket, ki implementira kompleksna števila in operacije nad njimi iz matematične knjižnjice Apache Commons [6].
Potence N-tega primitivnega korena enote izračunamo znova ob vsaki transformaciji. Za izračun smo uporabili Eulerjevo formulo (2.19):

$$\omega_N^k = e^{-i\frac{2\pi}{N}k} = \cos(\frac{2\pi}{N}k) - i\sin(\frac{2\pi}{N}k);\ k, N \in \mathbb{N}_0 \tag{3.1}$$

Teh vrednosti nismo izračunali že v naprej in jih tudi nismo shranili v kakršnokoli vrsto tabele za kasnejšo ponovno uporabo.
Prav tako vsi algoritmi uporabljajo enako funckijo za izračun DFT in izračun inverzne DFT. S pomočjo vhodnega parametra funkcije za izračun transformacije (boolean spremenljivka) določimo, če računamo inverzno transformacijo. V primeru računanja inverza se v zadnji fazi izvajanja vsaka izhodna vrednost deli še z dolžino vhodnih podatkov (N).

Izvorno kodo implementiranih algoritmov in celoten ALGator projekt smo shranili tudi v javno dosegljiv GitHub repozitorij [11].

3.1 Diskretna Fourierjeva transformacija

Definicija 3.1 *Diskretna Fourierjeva transformacija (DFT)* $\overline{X}_k$ *vhodnega zaporedja* x_k *dolžine* N *je podana z naslednjo enačbo [7]:*

$$\overline{X}_k = \sum_{r=0}^{N-1} x_r \omega_N^{rk};\ k = 0, 1,\ \ldots,\ N-1. \tag{3.2}$$

Kjer je ω_N *N-ti primitivni koren enote podan z enačbo* (2.27)*.*

Iz zgoraj navedene enačbe (3.2) razberemo, da je število kompleksnih seštevanj S in število kompleksnih množenj M v DFT sledeče:

$$S = N \cdot N = N^2 \tag{3.3}$$

in

$$M = N \cdot N = N^2. \tag{3.4}$$

Pri čemer nismo upoštevali izračuna potenc N-tega korena enote.

Ta algoritem smo implementirali striktno po zgornji enačbi (3.2) zato, da bomo lahko meritve ostalih algoritmov primerjali z definicijo DFT.

3.2 Osnova 2

Algoritem Cooley-Tukey (1965), ki sta ga razvila James William Cooley in John Tukey, razdeli problem velikosti N na več podproblemov (v primeru tega algoritma na dva), tako da velja $N = N_1 \cdot N_2$, kjer sta N_1 število podproblemov in N_2 velikost podproblemov. To nam omogoči, da izračun ene DFT razbijemo na več manjših DFT in zaradi posebnih lastnosti N-tega primitivnega korena enote, ki smo jih navedli v razdelku 2.3, zmanjšamo čas

izračuna. V našem primeru na $O(N \cdot \log N)$.
Predpostavimo, da želimo izračunati DFT $\overline{X}_k$ vhodnega zaporedja x_k dolžine N, kjer velja $N = 2^t; t \in \mathbb{N}$. Tedaj lahko izberemo za $N_1 = 2$ in $N_2 = 2^{t-1}$, kar pomeni, da smo razdelili vhodno množico podatkov na dve množici velikosti $\frac{N}{2}$. Ta način razdelitve vhodnih podatkov na dve podmnožici imenujemo algoritem z osnovo 2 (angl. radix-2). Predstavili bomo dve vrsti algoritma z osnovo 2. Razlikujeta se po načinu razdelitve vhodnih podatkov x_k v dve množici.
Glavno idejo algoritma z osnovo 2 smo razdelili na tri korake [8]:

1. korak: razdelimo problem na dva podproblema manjše velikosti.

2. korak: rešimo vsak podproblem z enakim algoritmom.

3. korak: sestavimo rešitev začetnega problema iz rešitve obeh podproblemov.

3.2.1 Decimacija v časovnem prostoru (DIT)

Če vhodno zaporedje x_k dolžine N razdelimo na sodi in lihi del x_{2m} ter x_{2m+1}, potem lahko DFT $\overline{X}_k$ zapišemo kot [7]:

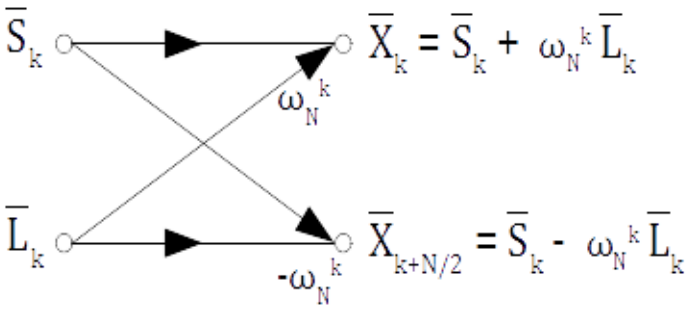
$$\overline{X}_k = \sum_{m=0}^{\frac{N}{2}-1} x_{2m}\omega_N^{2mk} + \omega_N^k \sum_{m=0}^{\frac{N}{2}-1} x_{2m+1}\omega_N^{2mk} \tag{3.5}$$

in, če upoštevamo še lastnost $\omega_N^{\frac{N}{2}} = -1$,

$$\overline{X}_{k+\frac{N}{2}} = \sum_{m=0}^{\frac{N}{2}-1} x_{2m}\omega_N^{2mk} - \omega_N^k \sum_{m=0}^{\frac{N}{2}-1} x_{2m+1}\omega_N^{2mk} \tag{3.6}$$

$$k = 0, 1, \ \ldots, \ \frac{N}{2} - 1.$$

Če v zgornjih dveh enačbah (3.5) in (3.6) upoštevamo lastnost (d) N-tega primitivnega korena enote, opazimo, da sta obe vsoti sodih x_{2m} ter lihih



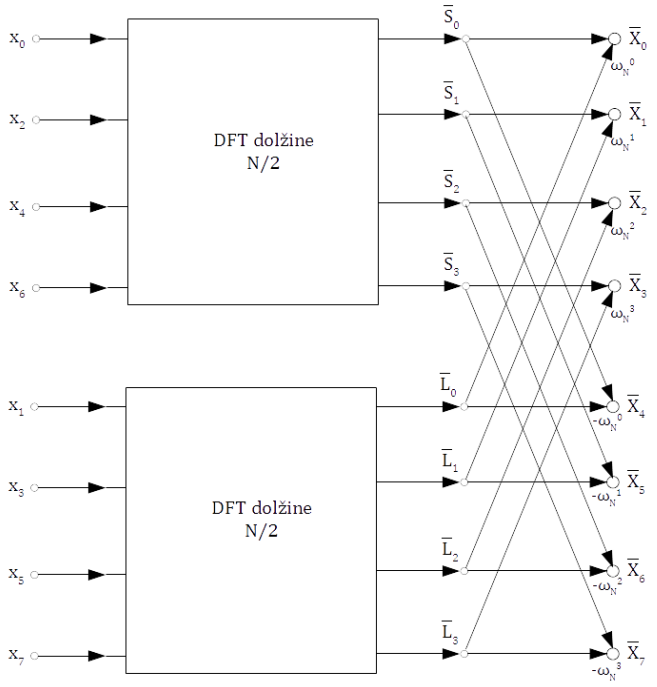
Slika 3.1: Metuljček Cooley-Tukey.

x_{2m+1} zopet DFT dolžine $\frac{N}{2}$. Če DFT sodih x_{2m} označimo s $\overline{S}_k$ in DFT lihih x_{2m+1} z $\overline{L}_k$, dobimo:

$$\overline{X}_k = \overline{S}_k + \omega_N^k \overline{L}_k \tag{3.7}$$

$$\overline{X}_{k+\frac{N}{2}} = \overline{S}_k - \omega_N^k \overline{L}_k \tag{3.8}$$

Izračun, ki je zapisan z zgornjima enačbama (3.7) in (3.8), predstavljen v spodnji sliki 3.1, je v literaturi pogosto omenjen kot *metuljček Cooley-Tukey* (angl. Cooley-Tukey butterfly).



Slika 3.2: Zadnji korak računanja DFT z decimacijo v časovnem prostoru (N=8).

Takšen pristop računanja DFT imenujemo *decimacija v časovnem prostoru* (angl. decimation in time). Na takšen način smo izračun DFT na N točkah nadomestili z dvema DFT dolžine $\frac{N}{2}$ in N dodatnimi seštevanji in $\frac{N}{2}$ množenji z ω_N^k. Enak postopek potem uporabimo, da nadomestimo vsako DFT dolžine $\frac{N}{2}$ še z dodatnima dvema DFT dolžine $\frac{N}{4}$, in N dodatnimi seštevanji in $\frac{N}{2}$ množenji.

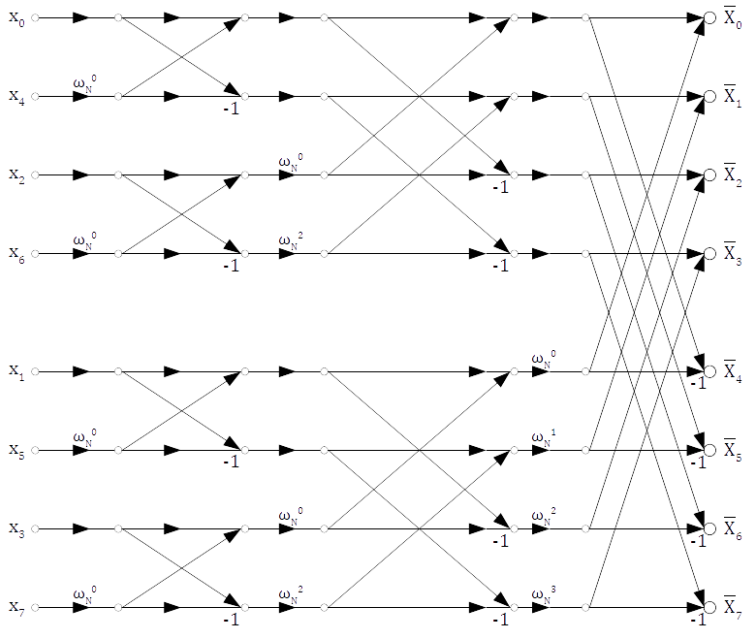
Sistematična uporaba te metode izračuna DFT dolžine 2^t v $t = \log_2 N$ korakih. Vsak korak razdeli 2^i diskretnih Fourierjevih transformacij dolžine 2^{t-i} v 2^{i+1} diskretnih Fourierjevih transformacij dolžine 2^{t-i-1} in pridela N dodatnih seštevanj in $\frac{N}{2}$ množenj. Posledično je število kompleksnih množenj M in število kompleksnih seštevanj S, ki jih zahteva izračun DFT dolžine N z

algoritmom osnove 2, enako:

$$M = \frac{N}{2} \log_2 N \tag{3.9}$$

in

$$S = N \log_2 N. \tag{3.10}$$



Slika 3.3: Algoritem z osnovo 2 (decimacija v časovnem prostoru, N=8).

3.2.2 Decimacija v frekvenčnem prostoru (DIF)

Drugo obliko algoritma z osnovo 2 dobimo, če vhodno zaporedje x_k dolžine N enostavno razdelimo na prvo in drugo polovico. Tako dobimo dve podmnožici x_k in $x_{k+\frac{N}{2}}$ velikosti $\frac{N}{2}$. Na takšen način, ki ga imenujemo *decimacija v frekvenčnem prostoru* (angl. decimation in frequency), zapišemo Fourierjevo transformacijo z enačbo [7]:

$$\overline{X}_k = \sum_{m=0}^{\frac{N}{2}-1} (x_m + \omega_N^{k\frac{N}{2}} x_{m+\frac{N}{2}}) \omega_N^{mk} \tag{3.11}$$

Sedaj lahko sode in lihe vrednosti $\overline{X}_k$ izračunamo ločeno. Za sode $\overline{X}_k$ k nadomestimo z $2k$ in dobimo [7]:

$$\overline{X}_{2k} = \sum_{m=0}^{\frac{N}{2}-1} (x_m + x_{m+\frac{N}{2}})\omega_N^{2mk} \tag{3.12}$$

$$k = 0, 1, \ \ldots, \ \frac{N}{2} - 1.$$

Pri lihih $\overline{X}_k$ pa k nadomestimo z $2k+1$, potem dobimo [7]:

$$\overline{X}_{2k+1} = \sum_{m=0}^{\frac{N}{2}-1} (x_m - x_{m+\frac{N}{2}})\omega_N^m\omega_N^{2mk} \tag{3.13}$$

$$k = 0, 1, \ \ldots, \ \frac{N}{2} - 1.$$

Sedaj v zgornjih dveh enačbah (3.12) in (3.13) označimo:

$$p_m = (x_m + x_{m+\frac{N}{2}}) \tag{3.14}$$

in

$$d_m = (x_m - x_{m+\frac{N}{2}})\omega_N^m, \tag{3.15}$$

potem dobimo bolj pregledni obliki enačbe za izračun sodih $\overline{X}_{2k}$ in lihih $\overline{X}_{2k+1}$ elementov DFT:

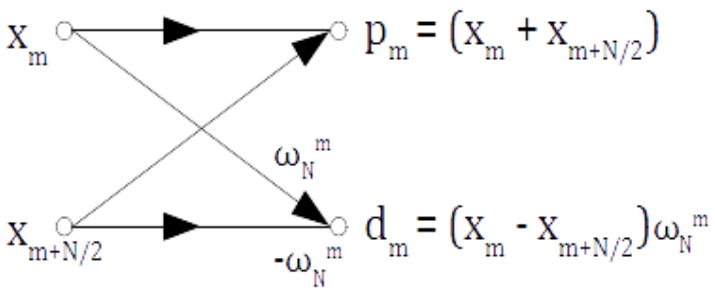
$$\overline{X}_{2k} = \sum_{m=0}^{\frac{N}{2}-1} p_m\omega_N^{2mk} \tag{3.16}$$

$$\overline{X}_{2k+1} = \sum_{m=0}^{\frac{N}{2}-1} d_m\omega_N^{2mk} \tag{3.17}$$

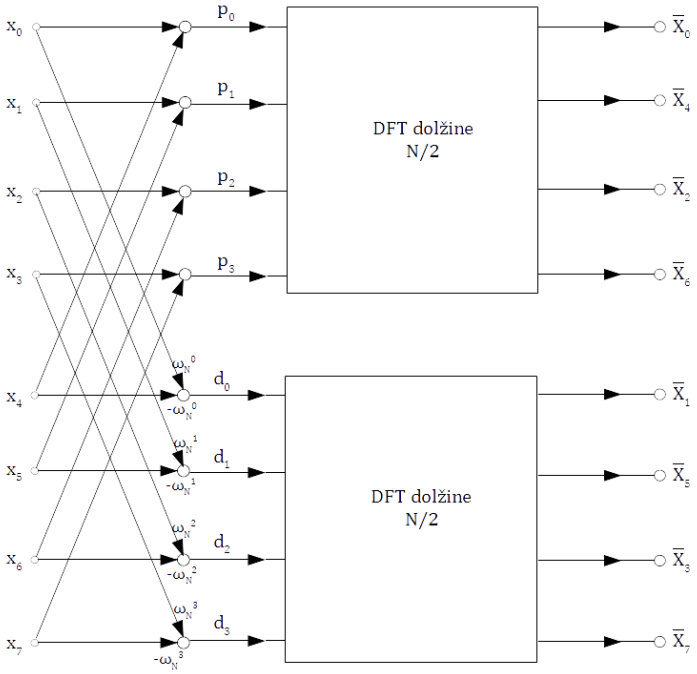
$$k = 0, 1, \ \ldots, \ \frac{N}{2} - 1.$$

Izračun, ki je zapisan z zgornjima enačbama (3.14) in (3.15), predstavljen v spodnji sliki 3.4, je v literaturi pogosto omenjen kot *metuljček Gentleman-Sande* (angl. Gentleman-Sande butterfly). Poimenovan po W. M. Gentleman

in G. Sande (1966), ki sta neodvisno predstavila algoritem z osnovo 2 po metodi DIF [8].



Slika 3.4: Metuljček Gentleman-Sande.



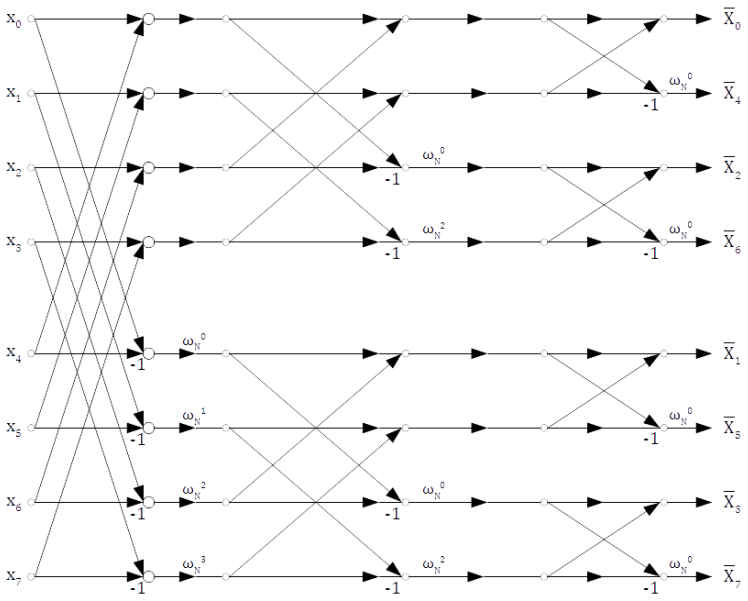
Slika 3.5: Prvi korak računanja DFT z decimacijo v frekvenčnem prostoru (N=8).

Na tak način smo zopet razdelili izračun večje DFT dolžine N na dve manjši DFT dolžine $\frac{N}{2}$, za to pa potrebujemo N kompleksnih seštevanj in $\frac{N}{2}$ kompleksnih množenj. Tako kot pri algoritmu z osnovo dva, ki uporablja decimacijo v časovnem prostoru, lahko tudi s tem načinom izračunamo DFT v $\log_2 N$ korakih. Vsak korak razdeli 2^i diskretnih Fourierjevih transformacij dolžine 2^{t-i} v 2^{i+1} diskretnih Fourierjevih transformacij dolžine 2^{t-i-1} z N dodatnimi kompleksnimi seštevanji in $\frac{N}{2}$ kompleksnimi množenji. To pomeni, da ta pristop, *decimacija v frekvenčnem prostoru*, potrebuje enako kompleksnih seštevanj in množenj kot pa alogritem z enako osnovo in decimacijo v časovnem prostoru. Če z M označimo število kompleksnih množenj, s S pa število kompleksnih seštevanj, to pomeni:

$$M = \frac{N}{2} \log_2 N \tag{3.18}$$

in

$$S = N \log_2 N. \tag{3.19}$$



Slika 3.6: Algoritem z osnovo 2 (decimacija v frekvenčnem prostoru, N=8).

3.3 Osnova 4

V tem razdelku bomo predpostavili, da obravnavamo DFT velikosti $N = 4^t = 2^{2t}$. Opazimo, da lahko DFT takšne velikosti izračunamo tudi z algoritmom z osnovo dva, predstavljenim v prejšnjem razdelku 3.2. Ta algoritem bomo obravnavali zato, ker ima manj aritmetičnih operacij kot algoritem z osnovo dva.

Algoritem z osnovo štiri je še ena izmed mnogih variacij algoritma Cooley-Tukey. Če se spomnimo iz razdelka 3.2, je glavna ideja, da nadomestimo izračun ene DFT z več manjšimi, tako da velja $N = N_1 \cdot N_2$. V tem primeru bomo izbrali $N_1 = 4$ in $N_2 = 2^{t-2}$. To je enako, kot če bi razbili vhodno zaporednje x_m v 4 manjša zaporednja velikosti $\frac{N}{4}$. Poznamo več vrst algoritma z osnovo 4. Med seboj se razlikujejo predvsem po tem, kako razdelijo vhodne podatke x_m na manjša zaporedja.

3.3.1 Decimacija v časovnem prostoru (DIT)

Če razdelimo vhodno zaporenje x_m na štiri manjša zaporedja x_{4m}, x_{4m+1}, x_{4m+2}, x_{4m+3} velikosti $\frac{N}{4}$, kjer $m = 0, 1, \ldots, \frac{N}{4} - 1$, potem govorimo o *decimaciji v časovnem prostoru* (angl. decimation in time). Pri takšni razdelitvi lahko potem osnovno enačbo DFT (3.2) zapišemo z delnimi vsotami [8]:

$$\overline{X}_k = \sum_{r=0}^{\frac{N}{4}-1} x_{4r}\omega_N^{k(4r)} + \sum_{r=0}^{\frac{N}{4}-1} x_{4r+1}\omega_N^{k(4r+1)} + \sum_{r=0}^{\frac{N}{4}-1} x_{4r+2}\omega_N^{k(4r+2)} + \sum_{r=0}^{\frac{N}{4}-1} x_{4r+3}\omega_N^{k(4r+3)} \tag{3.20}$$

$$\overline{X}_k = \sum_{r=0}^{\frac{N}{4}-1} x_{4r}\omega_N^{k(4r)} + \omega_N^k \sum_{r=0}^{\frac{N}{4}-1} x_{4r+1}\omega_N^{k(4r)} + \omega_N^{2k} \sum_{r=0}^{\frac{N}{4}-1} x_{4r+2}\omega_N^{k(4r)} + \omega_N^{3k} \sum_{r=0}^{\frac{N}{4}-1} x_{4r+3}\omega_N^{k(4r)} \tag{3.21}$$

$$\overline{X}_k = \sum_{l=0}^{3} \omega_N^{lk} \sum_{r=0}^{\frac{N}{4}-1} x_{4r+l}\ \omega_N^{4rk} \tag{3.22}$$

$$k = 0, 1, \ \ldots, \ N - 1.$$

Ko smo razdelili vhodno zaporedje na štiri manjša zaporednja: $\{y_r | y_r = x_{4r},\ 0 \leq r \leq \frac{N}{4} - 1\}$, $\{z_r | z_r = x_{4r+1},\ 0 \leq r \leq \frac{N}{4} - 1\}$, $\{g_r | g_r = x_{4r+2},\ 0 \leq r \leq \frac{N}{4} - 1\}$ in $\{h_r | h_r = x_{4r+3},\ 0 \leq r \leq \frac{N}{4} - 1\}$, potem lahko podprobleme velikosti $\frac{N}{4}$ s pomočjo lastnosti (d) N-tega primitivnega korena enote definiramo kot [8]:

$$\overline{Y}_k = \sum_{r=0}^{\frac{N}{4}-1} y_r \omega_{\frac{N}{4}}^{kr} \tag{3.23}$$

$$\overline{Z}_k = \sum_{r=0}^{\frac{N}{4}-1} z_r \omega_{\frac{N}{4}}^{kr} \tag{3.24}$$

$$\overline{G}_k = \sum_{r=0}^{\frac{N}{4}-1} g_r \omega_{\frac{N}{4}}^{kr} \tag{3.25}$$

$$\overline{H}_k = \sum_{r=0}^{\frac{N}{4}-1} h_r \omega_{\frac{N}{4}}^{kr} \tag{3.26}$$

$$k = 0, 1, \ \ldots, \ \frac{N}{4} - 1.$$

Vse, kar moramo sedaj storiti je, da te podprobleme rešimo na enak način (z metodo deli in vladaj) na koncu pa rešitev vhodnega problema dobimo z upoštevanjem enačbe (3.22) in posebnimi lastnostmi N-tega primitivnega korena enote $\omega_N^{\frac{N}{4}} = -i$, $\omega_N^{\frac{N}{2}} = -1$ in $\omega_N^{3\frac{N}{4}} = i$. Tako lahko rešitev $\overline{X}_k$ zapišemo s pomočjo $\overline{Y}_k$, $\overline{Z}_k$, $\overline{G}_k$ *in* $\overline{H}_k$ za vse $k = 0, 1, \ \ldots, \ \frac{N}{4} - 1$ [8]:

$$\overline{X}_k = \overline{Y}_k + \omega_N^k \overline{Z}_k + \omega_N^{2k} \overline{G}_k + \omega_N^{3k} \overline{H}_k, \tag{3.27}$$

$$\overline{X}_{k+\frac{N}{4}} = \overline{Y}_k - i\omega_N^k \overline{Z}_k - \omega_N^{2k} \overline{G}_k + i\omega_N^{3k} \overline{H}_k, \tag{3.28}$$

$$\overline{X}_{k+\frac{N}{2}} = \overline{Y}_k - \omega_N^k \overline{Z}_k + \omega_N^{2k} \overline{G}_k - \omega_N^{3k} \overline{H}_k, \tag{3.29}$$

$$\overline{X}_{k+3\frac{N}{4}} = \overline{Y}_k + i\omega_N^k \overline{Z}_k - \omega_N^{2k} \overline{G}_k - i\omega_N^{3k} \overline{H}_k. \tag{3.30}$$

Če bi algoritem z osnovo 4 implementirali po zgornjih enačbah (3.27), (3.28), (3.29) in (3.30) potem bi en korak tega algoritma z osnovo 4 potreboval več aritmetičnih operacij kot pa dva koraka algoritma z osnovo 2, ker bi nekatere delne rezultate izračunali večkrat. Ampak, če prepoznamo te delne rezultate in jih izračunamo samo enkrat, potem bo en korak algoritma z osnovo 4 zahteval manj aritmetičnih operacij kot dva koraka algoritma z osnovo 2, tako bomo zmanjšali časovno zahtevnost alogritma z osnovo 4. Štiri prepoznane delne rezultate bomo zapisali spodaj znotraj oklepajev [8]:

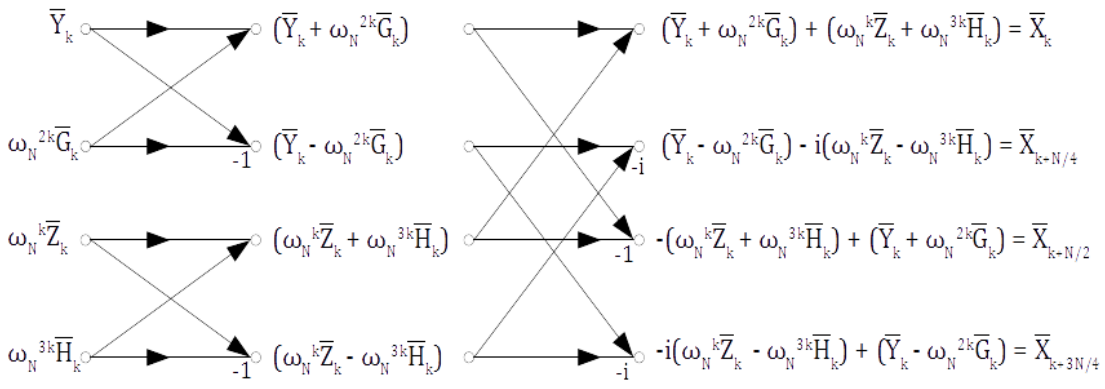
$$\overline{X}_k = (\overline{Y}_k + \omega_N^{2k} \overline{G}_k) + (\omega_N^k \overline{Z}_k + \omega_N^{3k} \overline{H}_k), \tag{3.31}$$

$$\overline{X}_{k+\frac{N}{4}} = (\overline{Y}_k - \omega_N^{2k} \overline{G}_k) - i(\omega_N^k \overline{Z}_k - \omega_N^{3k} \overline{H}_k), \tag{3.32}$$

$$\overline{X}_{k+\frac{N}{2}} = -(\omega_N^k \overline{Z}_k + \omega_N^{3k} \overline{H}_k) + (\overline{Y}_k + \omega_N^{2k} \overline{G}_k), \tag{3.33}$$

$$\overline{X}_{k+3\frac{N}{4}} = i(\omega_N^k \overline{Z}_k - \omega_N^{3k} \overline{H}_k) + (\overline{Y}_k - \omega_N^{2k} \overline{G}_k), \tag{3.34}$$

kjer $k = 0, 1, \ldots, \frac{N}{4} - 1$. Izračun zapisan z enačbami (3.31), (3.32), (3.33) in (3.34) lahko predstavimo z dvostopenjskim metuljčkom (slika 3.7).



Slika 3.7: Dvostopenjski metuljček za algoritem z osnovo 4 (DIT).

Opazimo, da morajo biti $\omega_N^{2k}\overline{G}_k$, $\omega_N^{k}\overline{Z}_k$ in $\omega_N^{3k}\overline{H}_k$ izračunani pred štirimi delnimi rezultati. Ker je velikost vsakega podproblema $\frac{N}{4}$, potrebujemo $3\frac{N}{4}$ kompleksnih množenj in N kompleksnih seštevanj v prvi stopnji. V drugi stopnji, ker ni množenj z N-timi primitivnimi koreni enote, potrebujemo samo N kompleksnih seštevanj. Posledično je število kompleksnih množenj M in število kompleksnih seštevanj S, ki jih zahteva izračun DFT dolžine N z algoritmom (DIT) osnove 4, enako:

$$M = 3\frac{N}{4}\log_4 N \tag{3.35}$$

in

$$S = 2N\log_4 N. \tag{3.36}$$

3.3.2 Decimacija v frekvenčnem prostoru (DIF)

Algoritem z osnovo 4 z *decimacijo v frekvenčnem prostoru* (angl. decimation in frequency) dobimo tako, da zaporedje v frekvenčni domeni razdelimo na štiri manjša zaporedja, označena z $\overline{Y}_k = \overline{X}_{4k}$ za $0 \leq k \leq \frac{N}{4} - 1$, $\overline{Z}_k = \overline{X}_{4k+1}$ za $0 \leq k \leq \frac{N}{4} - 1$, $\overline{G}_k = \overline{X}_{4k+2}$ za $0 \leq k \leq \frac{N}{4} - 1$ in $\overline{H}_k = \overline{X}_{4k+3}$ za $0 \leq k \leq \frac{N}{4} - 1$. Takšno razdelitev smo zopet dobili s pomočjo osnovne

enačbe DFT (3.2) [8]:

$$\overline{X}_r = \sum_{l=0}^{\frac{N}{4}-1} x_l \omega_N^{rl} + \sum_{l=\frac{N}{4}}^{\frac{N}{2}-1} x_l \omega_N^{rl} + \sum_{l=\frac{N}{2}}^{\frac{3N}{4}-1} x_l \omega_N^{rl} + \sum_{l=\frac{3N}{4}}^{N-1} x_l \omega_N^{rl} \tag{3.37}$$

$$\overline{X}_r = \sum_{l=0}^{\frac{N}{4}-1} x_l \omega_N^{rl} + \sum_{l=0}^{\frac{N}{4}-1} x_{l+\frac{N}{4}} \omega_N^{r(l+\frac{N}{4})} + \sum_{l=0}^{\frac{N}{4}-1} x_{l+\frac{N}{2}} \omega_N^{r(l+\frac{N}{2})} + \sum_{l=0}^{\frac{N}{4}-1} x_{l+\frac{3N}{4}} \omega_N^{r(l+\frac{3N}{4})} \tag{3.38}$$

$$\overline{X}_r = \sum_{l=0}^{\frac{N}{4}-1} x_l \omega_N^{rl} + \omega_4^r \sum_{l=0}^{\frac{N}{4}-1} x_{l+\frac{N}{4}} \omega_N^{rl} + \omega_4^{2r} \sum_{l=0}^{\frac{N}{4}-1} x_{l+\frac{N}{2}} \omega_N^{rl} + \omega_4^{3r} \sum_{l=0}^{\frac{N}{4}-1} x_{l+\frac{3N}{4}} \omega_N^{rl} \tag{3.39}$$

$$\overline{X}_r = \sum_{l=0}^{\frac{N}{4}-1} (x_l + x_{l+\frac{N}{4}} \omega_4^r + x_{l+\frac{N}{2}} \omega_4^{2r} + x_{l+\frac{3N}{4}} \omega_4^{3r}) \omega_N^{rl}. \tag{3.40}$$

$$r = 0, 1, \ \ldots, \ N-1.$$

Štiri manjša zaporedja bomo dobili, če nadomestimo $r = 4k$, $r = 4k+1$, $r = 4k+2$ in $r = 4k+3$ v zgornji enačbi (3.40) [8].

$$\begin{aligned}
\overline{Y}_k = \overline{X}_{4k} &= \sum_{l=0}^{\frac{N}{4}-1} (x_l + x_{l+\frac{N}{4}} \omega_4^{4k} + x_{l+\frac{N}{4}} \omega_4^{2\times 4k} + x_{l+\frac{3N}{4}} \omega_4^{3\times 4k}) \omega_N^{4kl} \\
&= \sum_{l=0}^{\frac{N}{4}-1} (x_l + x_{l+\frac{N}{4}} + x_{l+\frac{N}{2}} + x_{l+\frac{3N}{4}}) \omega_{\frac{N}{4}}^{kl} \\
&= \sum_{l=0}^{\frac{N}{4}-1} ((x_l + x_{l+\frac{N}{2}}) + (x_{l+\frac{N}{4}} + x_{l+\frac{3N}{4}})) \omega_{\frac{N}{4}}^{kl} \\
&= \sum_{l=0}^{\frac{N}{4}-1} y_l \omega_{\frac{N}{4}}^{kl}, \ k = 0, 1, \ \ldots, \ \frac{N}{4} - 1.
\end{aligned} \tag{3.41}$$

$$\begin{aligned}\overline{Z}_k = \overline{X}_{4k+1} &= \sum_{l=0}^{\frac{N}{4}-1} (x_l + x_{l+\frac{N}{4}}\omega_4^{4k+1} + x_{l+\frac{N}{4}}\omega_4^{2(4k+1)} + x_{l+\frac{3N}{4}}\omega_4^{3(4k+1)})\omega_N^{(4k+1)l} \\ &= \sum_{l=0}^{\frac{N}{4}-1} ((x_l - x_{l+\frac{N}{2}}) - i(x_{l+\frac{N}{4}} - x_{l+\frac{3N}{4}}))\omega_N^l \omega_{\frac{N}{4}}^{kl} \\ &= \sum_{l=0}^{\frac{N}{4}-1} z_l \omega_{\frac{N}{4}}^{kl},\ k = 0, 1,\ \ldots,\ \frac{N}{4} - 1. \end{aligned} \tag{3.42}$$

$$\begin{aligned}\overline{G}_k = \overline{X}_{4k+2} &= \sum_{l=0}^{\frac{N}{4}-1} (x_l + x_{l+\frac{N}{4}}\omega_4^{4k+2} + x_{l+\frac{N}{4}}\omega_4^{2(4k+2)} + x_{l+\frac{3N}{4}}\omega_4^{3(4k+2)})\omega_N^{(4k+2)l} \\ &= \sum_{l=0}^{\frac{N}{4}-1} ((x_l + x_{l+\frac{N}{2}}) - (x_{l+\frac{N}{4}} + x_{l+\frac{3N}{4}}))\omega_N^{2l} \omega_{\frac{N}{4}}^{kl} \\ &= \sum_{l=0}^{\frac{N}{4}-1} g_l \omega_{\frac{N}{4}}^{kl},\ k = 0, 1,\ \ldots,\ \frac{N}{4} - 1. \end{aligned} \tag{3.43}$$

$$\begin{aligned}\overline{H}_k = \overline{X}_{4k+3} &= \sum_{l=0}^{\frac{N}{4}-1} (x_l + x_{l+\frac{N}{4}}\omega_4^{4k+3} + x_{l+\frac{N}{4}}\omega_4^{2(4k+3)} + x_{l+\frac{3N}{4}}\omega_4^{3(4k+3)})\omega_N^{(4k+3)l} \\ &= \sum_{l=0}^{\frac{N}{4}-1} ((x_l - x_{l+\frac{N}{2}}) + i(x_{l+\frac{N}{4}} - x_{l+\frac{3N}{4}}))\omega_N^{3l} \omega_{\frac{N}{4}}^{kl} \\ &= \sum_{l=0}^{\frac{N}{4}-1} h_l \omega_{\frac{N}{4}}^{kl},\ k = 0, 1,\ \ldots,\ \frac{N}{4} - 1. \end{aligned} \tag{3.44}$$

Ker bomo za izračun štirih podproblemov uporabili dvostopenjski metuljček, smo v zgornjih enačbah delne vsote označili z:

$$y_k = (x_k + x_{k+\frac{N}{2}}) + (x_{k+\frac{N}{4}} + x_{k+\frac{3N}{4}}), \tag{3.45}$$

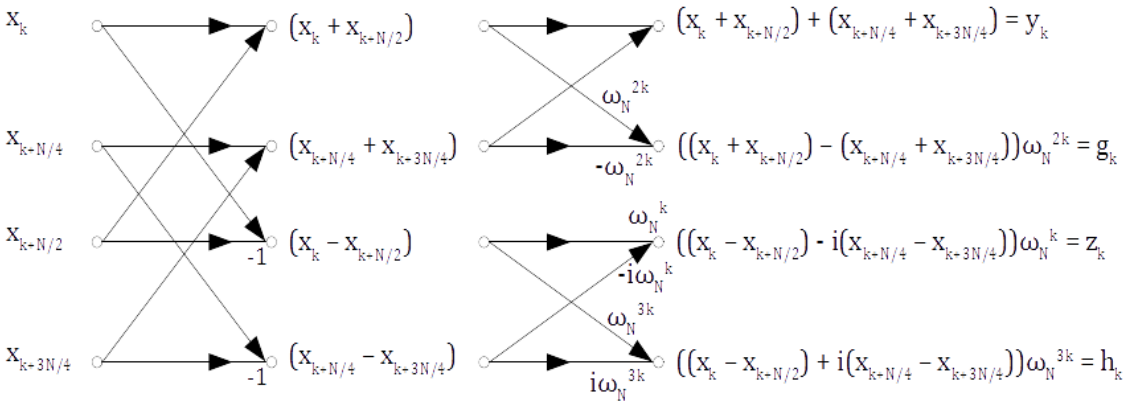
$$z_k = ((x_k - x_{k+\frac{N}{2}}) - i(x_{k+\frac{N}{4}} - x_{k+\frac{3N}{4}}))\omega_N^k, \tag{3.46}$$

$$g_k = ((x_k + x_{k+\frac{N}{2}}) - (x_{k+\frac{N}{4}} + x_{k+\frac{3N}{4}}))\omega_N^{2k}, \tag{3.47}$$

$$h_k = ((x_k - x_{k+\frac{N}{2}}) + i(x_{k+\frac{N}{4}} - x_{k+\frac{3N}{4}}))\omega_N^{3k}, \tag{3.48}$$

$$k = 0, 1, \ \ldots, \ \frac{N}{4} - 1.$$

Tako bomo lahko izračune zapisane v enačbah (3.45), (3.46), (3.47) in (3.48) zapisali z dvostopenjskim metuljčkom, kot smo prikazali na sliki 3.8.



Slika 3.8: Dvostopenjski metuljček za algoritem z osnovo 4 (DIF).

Če primerjamo DIT metuljček na sliki 3.7 in DIF metuljček na sliki 3.8, hitro opazimo, da je število kompleksnih množenj M in število kompleksnih steštevanj S enako kot pri decimaciji v časovnem prostoru (DIT), torej:

$$M = 3\frac{N}{4}\log_4 N \tag{3.49}$$

in

$$S = 2N\log_4 N. \tag{3.50}$$

3.4 Deljena osnova

Ker smo že predstavili algoritma z osnovo dva in osnovo štiri v razdelkih 3.2 in 3.3, bomo predstavili še algoritem z deljeno osnovo. Enako kot v prejšnjem razdelku o algoritmu z osnovo štiri 3.3 bomo tudi tukaj obravnavali DFT velikosti $N = 4^t = 2^{2t}$. Ta algoritem uporabi nekaj delnih vsot iz algoritma z osnovo 2 in nekaj iz algoritma z osnovo 4. Na ta način bomo tako še zmanjšali število kompleksnih množenj in seštevanj. Omenimo, da sta algoritem z deljeno osnovo prvič predstavila Duhamel in Hollman v letu 1984. Prav tako bomo zopet predstavili DIT in DIF implementaciji tega algoritma.

3.4.1 Decimacija v časovnem prostoru (DIT)

Algoritem z deljeno osnovo z *decimacijo v časovnem prostoru* (DIT) dobimo tako, da vhodno zaporedje x_k razdelimo na tri manjša zaporedja $\{y_k | y_k = x_{2k},\ 0 \leq r \leq \frac{N}{2}-1\}$, $\{z_k | z_k = x_{4k+1},\ 0 \leq r \leq \frac{N}{4}-1\}$ ter $\{h_k | h_k = x_{4k+3},\ 0 \leq r \leq \frac{N}{4}-1\}$. Ko na ta način razdelimo osnovno enačbo (3.2) DFT, dobimo [8]:

$$\begin{aligned} \overline{X}_r &= \sum_{l=0}^{N-1} x_l \omega_N^{rl} \\ &= \sum_{k=0}^{\frac{N}{2}-1} x_{2k} \omega_N^{r(2k)} + \sum_{k=0}^{\frac{N}{4}-1} x_{4k+1} \omega_N^{r(4k+1)} + \sum_{k=0}^{\frac{N}{4}-1} x_{4k+3} \omega_N^{r(4k+3)} \\ &= \sum_{k=0}^{\frac{N}{2}-1} x_{2k} \omega_N^{r(2k)} + \omega_N^r \sum_{k=0}^{\frac{N}{4}-1} x_{4k+1} \omega_N^{r(4k)} + \omega_N^{3r} \sum_{k=0}^{\frac{N}{4}-1} x_{4k+3} \omega_N^{r(4k)} \end{aligned} \tag{3.51}$$

$$r = 0, 1,\ \ldots,\ N-1.$$

Tako smo razdelili vhodni problem na tri manjše podprobleme, ki jih bomo rešili z enakim algoritmom. Označimo jih z [8]:

$$\overline{Y}_r = \sum_{k=0}^{\frac{N}{2}-1} x_{2k} \omega_N^{r(2k)} = \sum_{k=0}^{\frac{N}{2}-1} x_{2k} (\omega_N^2)^{rk} = \sum_{k=0}^{\frac{N}{2}-1} y_k \omega_{\frac{N}{2}}^{rk},\ r = 0, 1,\ \ldots,\ \frac{N}{2} - 1, \tag{3.52}$$

$$\overline{Z}_r = \sum_{k=0}^{\frac{N}{2}-1} x_{4k+1}\omega_N^{r(4k)} = \sum_{k=0}^{\frac{N}{4}-1} x_{4k+1}(\omega_N^4)^{rk} = \sum_{k=0}^{\frac{N}{4}-1} z_k\omega_{\frac{N}{4}}^{rk},\ r = 0, 1,\ \ldots,\ \frac{N}{4} - 1, \tag{3.53}$$

$$\overline{H}_r = \sum_{k=0}^{\frac{N}{2}-1} x_{4k+3}\omega_N^{r(4k)} = \sum_{k=0}^{\frac{N}{4}-1} x_{4k+3}(\omega_N^4)^{rk} = \sum_{k=0}^{\frac{N}{4}-1} h_k\omega_{\frac{N}{4}}^{rk},\ r = 0, 1,\ \ldots,\ \frac{N}{4} - 1. \tag{3.54}$$

Končno rešitev vhodnega problema velikosti N sestavimo po enačbi (3.51) za $r = 0, 1,\ \ldots,\ N-1$. Ker je $\overline{Y}_{r+k\frac{N}{2}} = \overline{Y}_r$ za $0 \leq r \leq \frac{N}{2} - 1$, $\overline{Z}_{r+k\frac{N}{4}} = \overline{Z}_r$ za $0 \leq r \leq \frac{N}{4}-1$ ter $\overline{H}_{r+k\frac{N}{4}} = \overline{H}_r$ za $0 \leq r \leq \frac{N}{4}-1$ lahko enačbo (3.51) zapišemo ponovno s pomočjo podproblemov $\overline{Y}_r$, $\overline{Y}_{r+\frac{N}{4}}$, $\overline{Z}_r$ *in* $\overline{H}_r$ za $0 \leq r \leq \frac{N}{4}-1$ [8].

$$\overline{X}_r = \overline{Y}_r + \omega_N^r\overline{Z}_r + \omega_N^{3r}\overline{H}_r = \overline{Y}_r + (\omega_N^r\overline{Z}_r + \omega_N^{3r}\overline{H}_r), \tag{3.55}$$

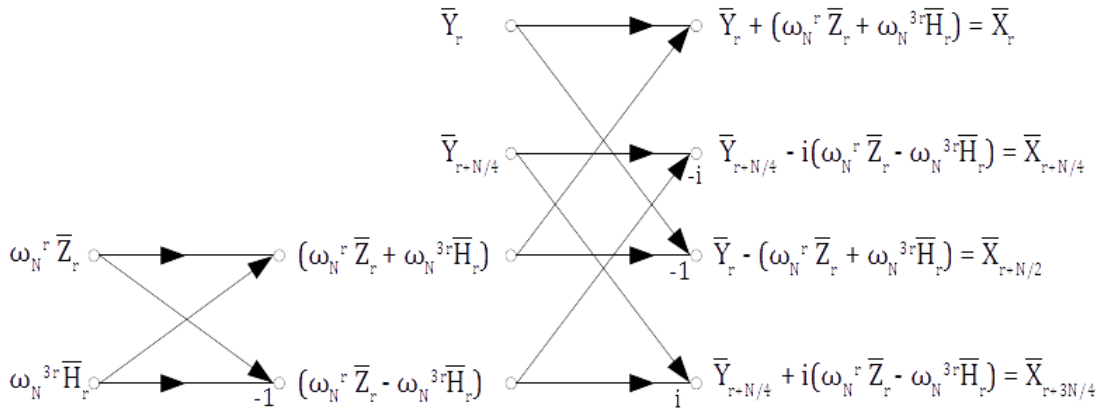
$$\overline{X}_{r+\frac{N}{4}} = \overline{Y}_{r+\frac{N}{4}} + \omega_N^{r+\frac{N}{4}}\overline{Z}_r + \omega_N^{3(r+\frac{N}{4})}\overline{H}_r = \overline{Y}_{r+\frac{N}{4}} - i(\omega_N^r\overline{Z}_r - \omega_N^{3r}\overline{H}_r), \tag{3.56}$$

$$\overline{X}_{r+\frac{N}{2}} = \overline{Y}_r + \omega_N^{r+\frac{N}{2}}\overline{Z}_r + \omega_N^{3(r+\frac{N}{2})}\overline{H}_r = \overline{Y}_r - (\omega_N^r\overline{Z}_r + \omega_N^{3r}\overline{H}_r), \tag{3.57}$$

$$\overline{X}_{r+\frac{3N}{4}} = \overline{Y}_{r+\frac{N}{4}} + \omega_N^{r+\frac{3N}{4}}\overline{Z}_r + \omega_N^{3(r+\frac{3N}{4})}\overline{H}_r = \overline{Y}_{r+\frac{N}{4}} + i(\omega_N^r\overline{Z}_r - \omega_N^{3r}\overline{H}_r), \tag{3.58}$$

$$r = 0, 1,\ \ldots,\ \frac{N}{4} - 1.$$

Izračune predstavljene v enačbah (3.55), (3.56), (3.57) ter (3.58) v literaturi imenujemo dvostopenjski *nesimetrični* DIT metuljček [8], ki je prikazan na sliki 3.9.



Slika 3.9: Dvostopenjski nesimetrični metuljček za algoritem z deljeno osnovo (DIT).

Opazimo, da smo v prvi stopnji metuljčka opravili dve kompleksni množenji in dve kompleksni seštevanji, v drugi stopnji pa štiri kompleksna seštevanja. Opomnimo še, da množenja z imaginarno enoto i ne štejemo, ker je vse kar naredi to, da zamenja realni in imaginarni del ter obrne predznak pri imaginarnem delu. Ta operacija je mnogo cenejša od običajnega seštevanja in množenja, zato je ne štejemo. Število potrebnih kompleksnih množenj M in število kompleksnih seštevanj S, ki jih potrebuje algoritem z deljeno osnovo z decimacijo v časovnem prostoru (DIT), je torej enako:

$$M = \frac{N}{2} \log_4 N \tag{3.59}$$

in

$$S = 3\frac{N}{2} \log_4 N. \tag{3.60}$$

3.4.2 Decimacija v frekvenčnem prostoru (DIF)

Algoritem z deljeno osnovo z *decimacijo v frekvenčnem prostoru* dobimo tako, da za reševanje podproblemov uporabimo algoritma z osnovo 2 in 4 z DIF. Glavni problem razdelimo na tri podprobleme $\overline{Y}_k = \overline{X}_{2k}$ za $0 \leq k \leq \frac{N}{2} - 1$,

$\overline{Z}_k = \overline{X}_{4k+1}$ za $0 \leq k \leq \frac{N}{4} - 1$ in $\overline{H}_k = \overline{X}_{4k+3}$ za $0 \leq k \leq \frac{N}{4} - 1$, kot bomo pokazali spodaj. Spomnimo se rešitve, ki smo jo razvili v algoritmu z osnovo 2 z DIF v (3.11), potem dobimo [8]:

$$\overline{X}_r = \sum_{m=0}^{\frac{N}{2}-1} x_m \omega_N^{rm} + \sum_{m=\frac{N}{2}}^{N-1} x_m \omega_N^{rm} = \sum_{m=0}^{\frac{N}{2}-1} (x_m + x_{m+\frac{N}{2}} \omega_N^{r\frac{N}{2}}) \omega_N^{rm}, \tag{3.61}$$

$$r = 0, 1, \ \ldots, \ N-1.$$

Če označimo $\overline{Y}_k = \overline{X}_{2k}$, $y_m = x_m + x_{m+\frac{N}{2}}$ potem prvi podproblem velikosti $\frac{N}{2}$ definiramo z:

$$\overline{Y}_k = \overline{X}_{2k} = \sum_{m=0}^{\frac{N}{2}-1} (x_m + x_{m+\frac{N}{2}}) \omega_N^{km} = \sum_{m=0}^{\frac{N}{2}-1} y_m \omega_N^{km}, \tag{3.62}$$

$$k = 0, 1, \ \ldots, \ \frac{N}{2} - 1.$$

Da pa bi dobili formulo za reševanje drugih dveh podproblemov velikosti $\frac{N}{4}$, se moramo spomniti enačbe iz definicije DFT (3.2), hkrati pa uporabiti enačbo, ki smo jo razvili v algoritmu z osnovo 4 z DIF v razdelku 3.3.2 [8].

$$\overline{X}_r = \sum_{m=0}^{N-1} x_m \omega_N^{rm} = \sum_{m=0}^{\frac{N}{4}-1} (x_m + x_{m+\frac{N}{4}} \omega_4^r + x_{m+\frac{N}{2}} \omega_4^{2r} + x_{m+\frac{3N}{4}} \omega_4^{3r}) \omega_N^{rm}, \tag{3.63}$$

$$r = 0, 1, \ \ldots, \ N-1.$$

Sedaj naredimo substituciji $r = 4k + 1$ in $r = 4k + 3$, da dobimo enačbi za druga dva podproblema:

$$\begin{aligned}
\overline{Z}_k &= \overline{X}_{4k+1} = \sum_{m=0}^{\frac{N}{4}-1} (x_m + x_{m+\frac{N}{4}} \omega_4^{4k+1} + x_{m+\frac{N}{2}} \omega_4^{2(4k+1)} + x_{m+\frac{3N}{4}} \omega_4^{3(4k+1)}) \omega_N^{(4k+1)m} \\
\overline{Z}_k &= \sum_{m=0}^{\frac{N}{4}-1} ((x_m - x_{m+\frac{N}{2}}) - i(x_{m+\frac{N}{4}} - x_{m+\frac{3N}{4}})) \omega_N^m \omega_{\frac{N}{4}}^{km} \\
\overline{Z}_k &= \sum_{m=0}^{\frac{N}{4}-1} z_m \omega_{\frac{N}{4}}^{km},
\end{aligned} \tag{3.64}$$

$$\begin{aligned}
\overline{H}_k &= \overline{X}_{4k+3} = \sum_{m=0}^{\frac{N}{4}-1} (x_m + x_{m+\frac{N}{4}}\omega_4^{4k+3} + x_{m+\frac{N}{2}}\omega_4^{2(4k+3)} + x_{m+\frac{3N}{4}}\omega_4^{3(4k+3)})\omega_N^{(4k+3)m} \\
\overline{H}_k &= \sum_{m=0}^{\frac{N}{4}-1} ((x_m - x_{m+\frac{N}{2}}) + i(x_{m+\frac{N}{4}} - x_{m+\frac{3N}{4}}))\omega_N^m \omega_{\frac{N}{4}}^{km} \\
\overline{H}_k &= \sum_{m=0}^{\frac{N}{4}-1} h_m \omega_{\frac{N}{4}}^{km}, \qquad (3.65)
\end{aligned}$$

$$k = 0, 1, \ \ldots, \ \frac{N}{4} - 1.$$

Ker bomo za izračun vseh podproblemov uporabili dvostopenjskega *nesimetričnega* metuljčka, smo v zgornjih enačbah delne vsote označili z:

$$y_m = (x_m + x_{m+\frac{N}{2}}), \qquad (3.66)$$

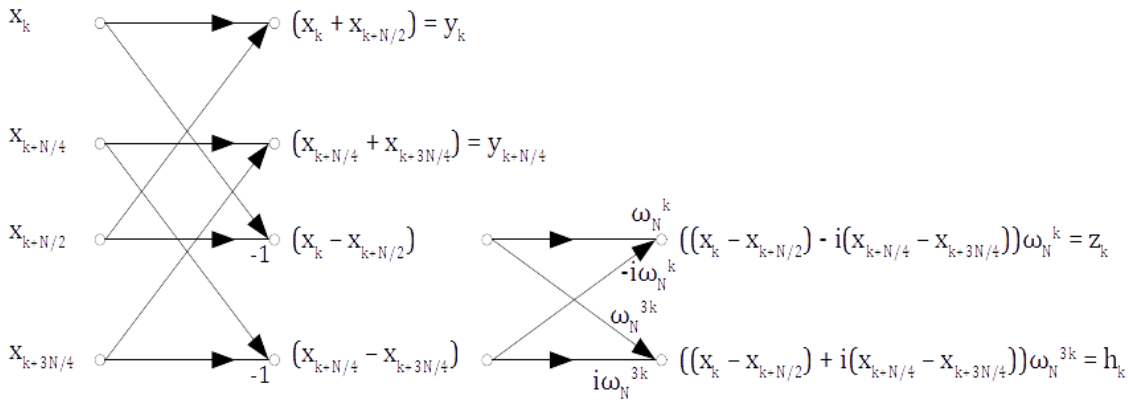
$$y_{m+\frac{N}{4}} = (x_{m+\frac{N}{4}} + x_{m+\frac{3N}{2}}), \qquad (3.67)$$

$$z_m = ((x_m - x_{m+\frac{N}{2}}) - i(x_{m+\frac{N}{4}} - x_{m+\frac{3N}{2}}))\omega_N^m, \qquad (3.68)$$

$$h_m = ((x_m - x_{m+\frac{N}{2}}) + i(x_{m+\frac{N}{4}} - x_{m+\frac{3N}{2}}))\omega_N^{3m}, \qquad (3.69)$$

$$m = 0, 1, \ \ldots, \ \frac{N}{4} - 1.$$

Tako bomo lahko izračune v enačbah (3.66), (3.67), (3.68) in (3.69) zapisali z dvostopenjskim nesimetričnim metuljčkom, ki je prikazan na sliki 3.10.



Slika 3.10: Dvostopenjski nesimetrični metuljček za algoritem z deljeno osnovo (DIF).

Opazimo, da je število kompleksnih množenj M in število kompleksnih seštevanj S enako kot pri decimaciji v časovnem prostoru (DIT), torej je število operacij v frekvenčnem prostoru (DIF) enako:

$$M = \frac{N}{2} \log_4 N \tag{3.70}$$

in

$$S = 3\frac{N}{2} \log_4 N. \tag{3.71}$$

3.5 Bluesteinov algoritem

Glede na to, da vsi predstavljeni algoritmi za izračun FFT v prejšnjih razdelkih zahtevajo, da je dolžina vhodnih podatkov potenca števila dva, nas je zanimalo, če obstaja kakšen algoritem, ki lahko izračuna DFT za poljubno dolžino vhodnih podatkov prej kot v času $\Theta(N^2)$, to je v $\Theta(N \log N)$. Predvsem za podatke, katerih dolžina je liho ali veliko praštevilo. To bi sicer lahko izračunali naprimer tudi z algoritmom z osnovo 2 tako, da bi dopolnili vhodno zaporedje z ničlami (0) do ustrezne dolžine, ki jo zahteva algoritem z osnovo 2. Vendar bi to v najslabšem primeru pomenilo, da bi dolžino vhodnih podatkov podvojili.

Predstavili bomo *Bluesteinov* algoritem, enega izmed mnogih (Raderjev [7], Bruunov [7], Berglandov [8], ...), ki lahko izračuna DFT podatkov, katerih velikost ni potenca števila dva. Zelo pomembno se nam zdi pripomniti, da ta algoritem nima nobenih omejitev glede dolžine vhodnih podatkov - N, prav tako ga tudi ni smiselno uporabiti za računanje DFT velikost $N = 2^t$, saj je algoritem z osnovo dva za takšne velikosti hitrejši.

Bluesteinov algoritem temelji na dveh idejah. Prva je spodnja identita 3.72:

$$rl = \frac{r^2}{2} + \frac{l^2}{2} - \frac{(r-l)}{2}, \tag{3.72}$$

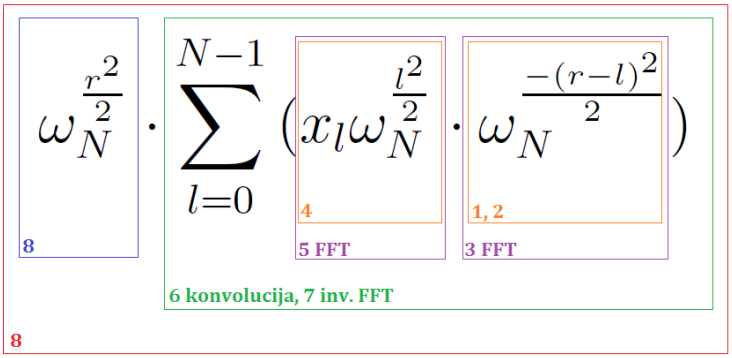
druga pa uporaba Chirp-Z transformacije in konvolucije za izračun DFT. Če uporabimo zgornjo enačbo (3.72) in rl v eksponentu N-tega primitivnega korena enote ω_N zapišemo kot:

$$\omega_N^{rl} = \omega_N^{\frac{r^2}{2}} \cdot \omega_N^{\frac{N^2}{2}} \cdot \omega_N^{\frac{-(r-l)^2}{2}}. \tag{3.73}$$

Potem lahko formulo za izračun DFT po definiciji 3.1 preuredimo v:

$$\overline{X}_r = \omega_N^{\frac{r^2}{2}} \cdot \sum_{l=0}^{N-1} (x_l \omega_N^{\frac{l^2}{2}} \cdot \omega_N^{\frac{-(r-l)^2}{2}}); \; r = 0, 1, \; \ldots, \; N-1. \tag{3.74}$$

Opazimo, da je zapis v oklepaju v zgornji enačbi (3.74) oblika konvolucije. N-ti primitivni koren enote $\omega_N^{\frac{r^2}{2}}$ pred vsoto pa ravno faktor za Chirp-Z transformacijo [8]. S spodnjo sliko 3.11 in kratkim opisom [8] bomo pojasnili Bluesteinov algoritem. Barvna števila na sliki se navezujejo na spodaj opisane in oštevilčene korake algoritma.



Slika 3.11: Oštevilčeni koraki Bluesteinovega algoritma.

- Izrazimo DFT produkt med matriko in vhodnim vektorjem v enačbi (3.2) kot produkt med simetrično Toeplitzovo matriko **H** in vhodnim vektorjem x.

- Razširitev Toeplitzove matrike **H** velikosti $N \times N$ v krožno matriko $\mathbf{H}^{(2)}$ velikosti $(2N-2) \times (2N-2)$. Če velja $2N-2 = 2^s$, potem bomo izračunali produkt med povečano krožno matriko in vektorjem. Če pa velja $2N-2 \neq 2^s$, potem bomo velikost krožne matrike povečali na $R = 2^s$. Na primer, če je N = 5, potem velja $2N-2 = 8 = 2^3$ in lahko izračunamo produkt med krožno matriko velikosti 8×8 in vhodnim vektorjem. Za primer, ko je N = 6, potem velja $2N-2 = 10 \neq 2^s$ in tedaj moramo velikost krožne matrike povečati na $R = 2^4 = 16$. To izvedemo zato, da lahko FT potrebne za konvolucijo izračunamo z algoritmom z osnovo 2.

- Diagonaliziramo matriko $\mathbf{H}^{(2)}$ s pomočjo identitete $\mathbf{\Omega H}^{(2)} = \mathbf{D\Omega}$ [8], kjer je $\mathbf{\Omega}$ matrika z M-timi primitivnimi koreni za izračun DFT dimenzije $M \times M$.

- Za poznane vhodne podatke x definiramo povečan vektor $\mathbf{y}^{(2)}$ dolžine R in izračunamo njegovo FT.

- Izvedemo konvolucijo med $\mathbf{y}^{(2)}$ in diagonalnimi elementi $\hat{h}_r$ diagonalne matrike $\mathbf{D}$. Nato izračunamo inverzno FT nad dobljenim rezultatom ter izluščimo transformirane vrednosti iz prvih N elementov.

Sedaj bomo po korakih opisali Bluestenov algoritem za izračun DFT [8]

$$\overline{X}_r = \sum_{l=0}^{N-1} x_l \omega_N^{rl},\ r\ =\ 0,\ 1,\ \ldots,\ N-1,\ N \neq 2^n. \tag{3.75}$$

1. korak: za dani N izračunajmo simetrično Toeplitzovo matriko $\mathbf{H}$:

$$h_l = \omega_N^{-\frac{1}{2}l^2},\ l\ =\ 0,\ 1,\ \ldots,\ N-1. \tag{3.76}$$

2. korak: razširimo $\mathbf{H}$ v krožno matriko $\mathbf{H}^{(2)}$: definirajmo R kot najmanjšo potenco števila dva, da velja $2^s = R \geq 2N-2$ in izračunajmo vektor $\mathbf{h}^{(2)}$ dolžine R definiran z:

$$h_l^{(2)} = h_l,\ l\ =\ 0,\ 1,\ \ldots,\ N-1, \tag{3.77}$$

$$h_l^{(2)} = h_{R-l},\ l\ =\ R-N+2,\ \ldots,\ R-1, \tag{3.78}$$

v primeru, ko velja $R > 2N-2$,

$$h_l^{(2)} = 0,\ l\ =\ N,\ \ldots,\ R-N+1. \tag{3.79}$$

3. korak: izračunamo diagonalo diagonalne matrike $\mathbf{D}$: uporabimo algoritem z osnovo dva, da izračunamo DFT produkta med matriko in vektorjem definiranega z:

$$\hat{h}_r = \sum_{l=0}^{R-1} h_l^{(2)} \omega_R^{rl},\ r\ =\ 0,\ 1,\ \ldots,\ R-1 \tag{3.80}$$

4. korak: za poznan x_l definirajmo povečan vektor $\mathbf{y}^{(2)}$ dolžine R kot:

$$y_l^{(2)} = x_l \omega_N^{\frac{1}{2}l^2},\ l\ =\ 0,\ 1,\ \ldots,\ N-1, \tag{3.81}$$

$$y_l^{(2)} = 0,\ l\ =\ N,\ \ldots,\ R-1. \tag{3.82}$$

5. korak: izračunamo FFT vektorja $\mathbf{y}^{(2)}$, imenujmo jo $\mathbf{\Omega y}^{(2)}$: za izračun uporabimo algoritem z osnovo dva:

$$\overline{Y}_r = \sum_{l=0}^{R-1} y_l^{(2)} \omega_R^{rl},\ r\ =\ 0,\ 1,\ \ldots,\ R-1. \tag{3.83}$$

6. korak: izračunamo $\hat{\boldsymbol{z}} = \mathbf{D}(\mathbf{\Omega y}^{(2)})$: pomnožimo $\overline{Y}_r$ s $\hat{h}$, kot bomo zapisali spodaj:

$$\hat{Z}_r = \hat{h}_r \overline{Y}_r,\ r\ =\ 0,\ 1,\ \ldots,\ R-1. \tag{3.84}$$

7. korak: izračunamo $\mathbf{z}^{(2)} = \mathbf{\Omega}^{-1}\hat{\boldsymbol{z}}$: uporabimo algoritem z osnovo dva za izračun inverzne DFT produkta med matriko in vektorjem definiranega z:

$$\overline{Z}_r = \frac{1}{R} \sum_{l=0}^{R-1} \hat{Z}_l \omega_R^{-rl},\ r\ =\ 0,\ 1,\ \ldots,\ R-1. \tag{3.85}$$

8. korak: izluščimo transformirane vrednosti $\overline{X}_r$ vhodnega zaporedja x_r iz prvih N elementov v vektorju $\mathbf{z}^{(2)}$ z:

$$\overline{X}_r = \overline{Z}_r \omega_N^{\frac{1}{2}r^2}, r\ =\ 0,\ 1,\ \ldots,\ N-1. \tag{3.86}$$

Opomnimo, da bi lahko implementirali algoritem na način, da bi se prvi trije koraki izvedli samo takrat, ko bi se spremenila velikost vhodnih podatkov - N. Vendar tega v naši implementaciji nismo storili, ker bomo merili hitrost algoritma vključno z vsemi koraki (ne glede na spremembo N).
Opazimo, da v tem algoritmu uporabimo dve DFT in eno inverzno DFT z osnovo dva nad podatki dolžine R. To pomeni, da bomo za izračun potrebovali 3-krat toliko operacij kompleksnih seštevanj in množenj kot pri algoritmu z osnovo dva, plus R kompleksnih množenj, ki jih pridelamo pri inverzni DFT. Upoštevati pa moramo seveda še število kompleksnih množenj v 4., 6. in 8.

koraku. Torej, če z $M2$ označimo število kompleksnih množenj v algoritmu z osnovo dva (3.49) in s $S2$ označimo število kompleksnih steštevanj v algoritmu z osnovo dva (3.50), potem lahko zapišemo skupno število kompleksnih množenj M in skupno število kompleksnih seštevanj S:

$$M = 3 \cdot M2 + R + N + R + N = 3 \cdot \frac{R}{2} \log_2 R + 2R + 2N \tag{3.87}$$

in

$$S = 3 \cdot S2 = 3 \cdot R \log_2 R. \tag{3.88}$$

Poglavje 4

Integracija v sistem ALGator

4.1 ALGator

ALGator [5] je sistem implementiran v programskem jeziku Java, ki razvijalcu algoritmov omogoča enostavno integracijo ter izvajanje algoritmov na izbranih testnih podatkih in analiziranje rezultatov izvajanj.
Sistem omogoča dodajanje in upravljanje s poljubnim številom projektov. V okviru enega projekta je definiran problem, testne množice vhodnih podatkov ter način reševanja nalog tega problema. Projekt lahko vsebuje poljubno število algoritmov, ki naloge rešujejo na predpisan način. Sistem omogoča analizo izvajanja posameznega algoritma ter primerjavo med algoritmi istega projekta [5].

4.2 Delovanje in struktura sistema

Sistem za svoje delovanje uporablja korenski imenik <*ALGator_root*> in njegove podimenike:

- *app*: imenik z zagonsko JAR datoteko in potrebnimi knjižnicami za delovanje sistema.

- *data_root*: imenik, ki vsebuje korenske imenike vseh projektov in datoteke potrebne za njihovo delovanje.

- *data_local*: imenik s testnimi množicami in prevedenimi (binarnimi) izvornimi datotekami vseh projektov.

Prav tako pa so znotraj posameznega projekta prisotne konfiguracijske datoteke tipa JSON in CSV, ki opisujejo vhodne ter izhodne parametre izvajanih algoritmov. Slika 4.1 prikazuje primer imenika <*data_root*>*/projects*, ki vsebuje definicije problemov v sistemu.

```
PROJ-ProjName           // project root folder
  proj                  // folder for all project files
    ProjName.atp        // project configuration file
    ProjName.atrd       // result description file
    src                 // folder for project source files
      [Project]AbsAlgorithm.java
      [Project]TestSetIterator.java
      [Project]TestCase.java
    bin                 // compiled classes of the project
      [Project]AbsAlgorithm.class
      [Project]TestSetIterator.class
      [Project]TestCase.class
    doc                 // html documentation for project
  algs                  // folder for algorithms
    ALG-AlgName         // algorithm root folder (*1)
      AlgName.atal      // algorithm configuration file
      src               // algorithm source files
        [Algorithm][Project]AbsAlgorithm.java
      bin               // compiled algorithm classes
        [Algorithm][Project]AbsAlgorithm.class
      doc               // html documentation for algorithm
      reports           // folder for algorithms reports


  tests                 // project-specific test files folder (*2)
    TestSetName1.atts   // one or more testset files
    doc                 // html documentation for testsets

  results               // folder with results (attr files)
  queries               // folder for predefined queries
  reports               // folder for projects reports
```

Slika 4.1: Primer imenika *<data_root>/projects* [5].

4.3 Opredelitev problema

Administrator v imeniku *<ALGator_root>/data_root/projects/ PROJ-<project_name>/* definira tri konfiguracijske datoteke:

- *<project_name>.atp*: vsebuje splošne informacije o projektu, kot so opis, avtor ter imena algoritmov za reševanje problema, testnih množic, javanskih datotek algoritma in pomožnih JAR datotek.

- *<project_name>-cnt.atrd*: konfiguracijska datoteka za meritve s števci. Na primer koda algoritma lahko vsebuje ukaz //@COUNT{<counter_name>, <value>}, ki poveča vrednost števca <counter_name> za <value>.

- *<project_name>-em.atrd*: konfiguracijska datoteka z opisi in tipi izhodnih ter časovnih indikatorjev.

Imenik vsebuje tudi podimenik *src*, v katerem najdemo naslednje tri javanske datoteke:

- *<project_name>TestCase.java*: razred, ki opisuje testni primer oziroma objekt s podatkovnimi strukturami, potrebnimi za shranjevanje podatkov o testnem primeru.

- *<project_name>TestSetIterator.java*: razred, ki prebere parametre posameznega testnega primera iz tekstovne datoteke in jih pretvori v objekt TestCase.java

- *<project_name>AbsAlgorithm.java*: abstraktni razred, katerega mora implementirati vsak razred algoritma za reševanje problema. V metodi *init()* pripravimo parametre, ki jih kasneje podamo metodi *run()*, kjer se izmeri čas izvajanja algoritma. Ko se izvajanje algoritma konča, se ob koncu izvede še metoda *done()*, v kateri iz rezultata algoritma preberemo še dodatne željene izhodne parametre.

4.4 Implementacija algoritmov

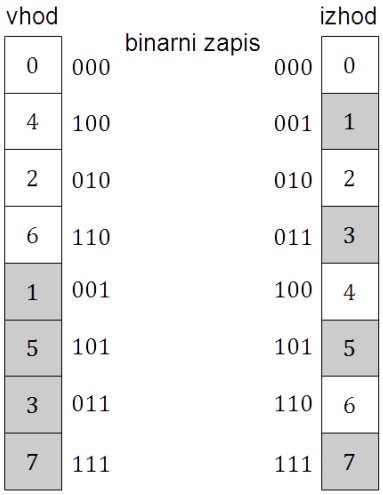
Algoritmi se nahajajo v imeniku *<ALGator_root>/data_root/projects/PROJ-<project_name>/algs/*. V tem imeniku ima vsak algoritem, ki rešuje dani problem svoj imenik z imenom *ALG-<algorithm_name>*. V njem se nahaja konfiguracijska datoteka, ki vsebuje informacije o posameznem algoritmu (npr. ime, avtor, naziv javanjske datoteke algoritma ipd.). Poleg konfiguracijske datoteke se tu nahaja še imenik *src*, ki vsebuje javansko datoteko *ALG-<algorithm_name>.java*

Implementirali smo algoritme, ki smo jih predstavili v poglavju 3, na več načinov. Poleg imena algoritma bomo v oklepaju navedli krajšo oznako, ki jo bomo uporabili tudi na grafih v naslednjem poglavju. In sicer:

1. Navadna diskretna Fourierjeva transformacija (DFT).
2. Bluesteinov algoritem.
3. Hitra Fourierjeva transformacija z osnovo 2 - decimacija v časovnem prostoru, implementiran na rekurziven način (ALG2T).
4. Hitra Fourierjeva transformacija z osnovo 2 - decimacija v časovnem prostoru, implementiran na iterativen način (ALG2TI).
5. Hitra Fourierjeva transformacija z osnovo 2 - decimacija v frekvenčnem prostoru, implementiran na rekurziven način (ALG2F).
6. Hitra Fourierjeva transformacija z osnovo 4 - decimacija v časovnem prostoru, implementiran na rekurziven način (ALG4T).
7. Hitra Fourierjeva transformacija z osnovo 4 - decimacija v časovnem prostoru, implementiran na iterativen način (ALG4TI).
8. Hitra Fourierjeva transformacija z osnovo 4 - decimacija v frekvenčnem prostoru, implementiran na rekurziven način (ALG4F).
9. Hitra Fourierjeva transformacija z deljeno osnovo (2 in 4) - decimacija v časovnem prosotru, implementiran na rekurziven način (ALG24T).
10. Hitra Fourierjeva transformacija z deljeno osnovo (2 in 4) - decimacija v frekvenčnem prosotru, implementiran na rekurziven način (ALG24F).

Z oznako ALG2 bomo označili vse algoritme z osnovo 2, z ALG4 vse z osnovo 4 ter z oznako ALG24 oba algoritma z deljeno osnovo.
Zaradi posebnih lastnosti Fourierjeve transformacije smo za izračun inverzne transformacije uporabili enako metodo kot za izračun transformacije. Edini razliki sta, da se pri izračunu inverza uporabijo konjugirane vrednosti potenc

N-tega primitivnega korena enote, lastnost (c) in v zadnji stopnji oziroma iteraciji vsak izhodni element delimo z dolžino vhodnih podatkov.
Pogoji za prekinitev rekurzije pri rekurzivnih algoritmih so odvisni od osnove. Rekurzija pri algoritmu z osnovo 2 in deljeno osnovo se zaključi, ko podproblem doseže velikost 2. Pri algoritmu z osnovo 4 pa, ko je velikost podproblema enaka 4. Ob vsakem klicu funkcije se ustvari nova tabela kompleksnih števil, v kateri vrnemo (delni) rezultat. Iterativni algoritmi izvedejo vse izračune s pomočjo ene tabele. Z zanko for znotraj funkcije iteriramo čez tabelo, tako kot je prikazano z metuljčki Cooley-Tukey na sliki 3.6. Opomnimo, da moramo pred računanjem v iterativnih algoritmih spremeniti vrsti red vhodnih podatkov, kot je prikazano na spodnji sliki 4.2.



Slika 4.2: Zamenjan vrstni red vhodnih podatkov (dolžine N = 8) pri iterativnem algoritmu.

4.5 Testni primeri

V imeniku *<ALGator_root>/data_root/projects/PROJ-<project_name>/tests/* se nahajajo množice testnih primerov in datoteke, ki opisujejo te množice. Torej je vsaka množica testnih primerov sestavljena iz ene datoteke z la-

snostmi, ki se imenuje *<set_name>.atts*, v kateri so zapisane lastnosti testnih primerov (npr. št. testnih primerov v datoteki, kratko ime, opis ipd.) in *<set_name>.txt*, v kateri so zapisani dejanski testni primeri. Vsaka vrstica v tej datoteki predstavlja en testni primer. Vsak testni primer pa je podan v naslednji obliki (vse skupaj v eni vrstici):
"*<test_name>:<length>:<min_allowed_error>:<type>:*
<input_values>:[<transformed_values>]".

- *<length>*: število kompleksnih števil v testnem primeru.
- *<min_allowed_error>*: da je transformacija označena za uspešno, mora biti razlika med dejansko in izračunano vrednostjo transformacije vsakega podanega števila manjša od tega parametra.
- *<type>*: definirali smo več tipov podajanja števil v testnih primerih. *IDENTICAL* - identični: vsa števila so enaka prvi podani vrednosti (parameter transformiranih vrednosti ni potreben, saj je rešitev trivialna zaradi posebnih lastnosti FT). *ALTERNATE* - alternirane vrednosti: vsa števila so enaka, le da smo števila na lihih pozicijah pomnožili z minus ena. Tudi pri tem tipu testnega primera ni potrebno podati transformiranih vrednosti, saj je rešitev zaradi posebnih lastnosti FT trivialna. *INLINE* - vhodne vrednosti smo podali z naštevanjem. Pri tem tipu pa je potrebno obvezno podati tudi transformirane vrednosti.

Vsa števila smo med seboj razmejili s presledkom (' '). Realno in kompleksno komponento posameznega števila po smo ločili s podpičjem (';').
Testne primere smo glede na dolžino vhodnih podatkov in velikost razdelili v šest testnih množic. Na primer dve izmed testnih množic vsebujeta samo testne primere, ki imajo liho število vhodnih podatkov, kar pomeni, da je transformacijo mogoče izračunati samo z navadno DFT ali z Bluesteinovim algoritmom. Potem naslednji dve testni množici vsebujeta testne primere z dolžino 2^N. Za nekatere primere v tej množici ni mogoče izračunati transformacije z algoritmi z osnovo štiri in deljeno osnovo. Ustvarili smo še dve

testni množici, v katerih so testni primeri dolžine 2^{2M}, za katere lahko transformacijo izračunamo z vsemi algoritmi, ki smo jih implementirali. Zadnja množica testnih primerov vsebuje teste s poljubnim številom vhodnih podatkov.
Minimalna dolžina testnega primera izmed vseh testov je 2, maksimalna pa 2^{18}.

4.6 Pomožni razredi

Pri implementaciji, delovanju in analiziranju algoritmov smo si pomagali tudi z drugimi razredi in knjižnicami, ki smo jih zapakirali v *FourierTransformAdmin.jar* datoteko. Ti razredi so:

- *Constants.java*: razred, v katerem so definirane vse konstante, ki jih projekt potrebuje za delovanje (npr. znak s katerim so delimitirana števila v testnih primerih ipd.).

- *UtilityFunctions.java*: razred s pomožnimi funkcijami, ki jih potrebujemo za delovanje projekta (npr. sortiranje tabel, izpis tabele v obliki niza ipd.).

- *AlgorithmError.java*: ta razred predstavlja objekt, v katerem so shranjena minimalna, maksimalna in povprečna napaka algoritma. Ta objekt ustvarimo v metodi *done()* v razredu *FourierTransformAbsAlgorithm.java* in ga uporabimo kot izhodni parameter.

- *commons-math3-3.6.jar [6]*: Apache commons matematična knjižnica, ki nam olajša delo s kompleksnimi števili. Tako nam ni bilo treba implementirati osnovnih operacij s kompleksnimi števili, kot so: seštevanje, odštevanje, množenje in deljenje.

Poglavje 5

Meritve

V tem poglavju bomo predstavili rezultate meritev, ki smo jih pridobili s pomočjo sistema ALGator. Predvsem nas je zanimal čas izvajanja v odvisnosti od velikosti vhodnih podatkov ter kakšne so razlike med rekurzivnimi in interativnimi implementacijami. Kot zanimivost in za primerjavo smo dodali tudi meritve navadne DFT. Prav tako bomo predstavili, kako število kompleksnih operacij in število rekurzivnih klicev vpliva na čas izvajanja.
Vse meritve smo izvedli na sistemu z naslednjimi specifikacijami:

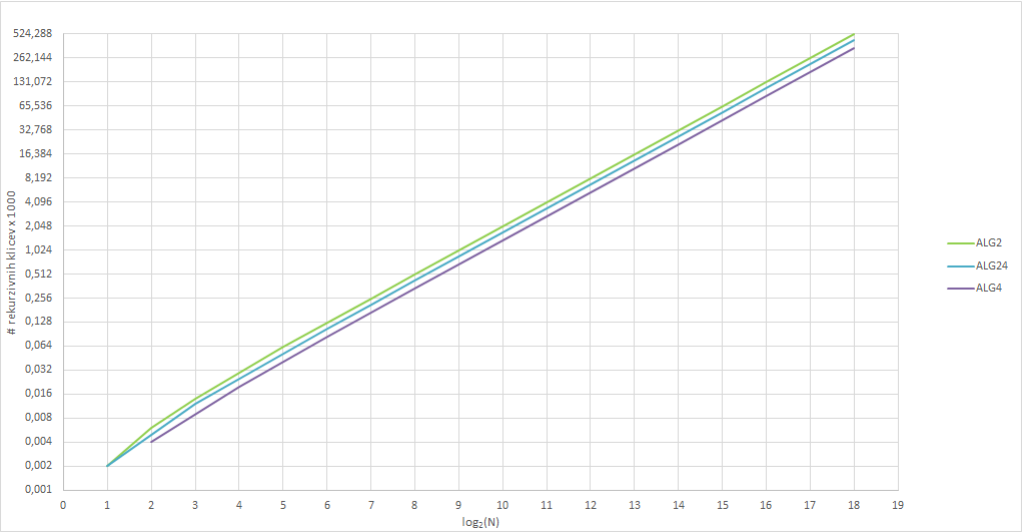
- procesor Intel Core i5 2.6 GHz z dvema jedroma,
- 10 GB delovnega pomnilnika,
- operacijski sistem Microsoft Windows 10 Proffesional.

Zaradi časovne zahtevnosti algoritma in posledično predolgega izvajanja, ki je implementiran po definiciji DFT, smo vrednosti zanj izmerili samo do velikosti vhodnih podatkov 16384 (2^{14}).
Vsi grafi v tem poglavju uporabljajo logaritmično skalo.

5.1 Rekurzivni klici

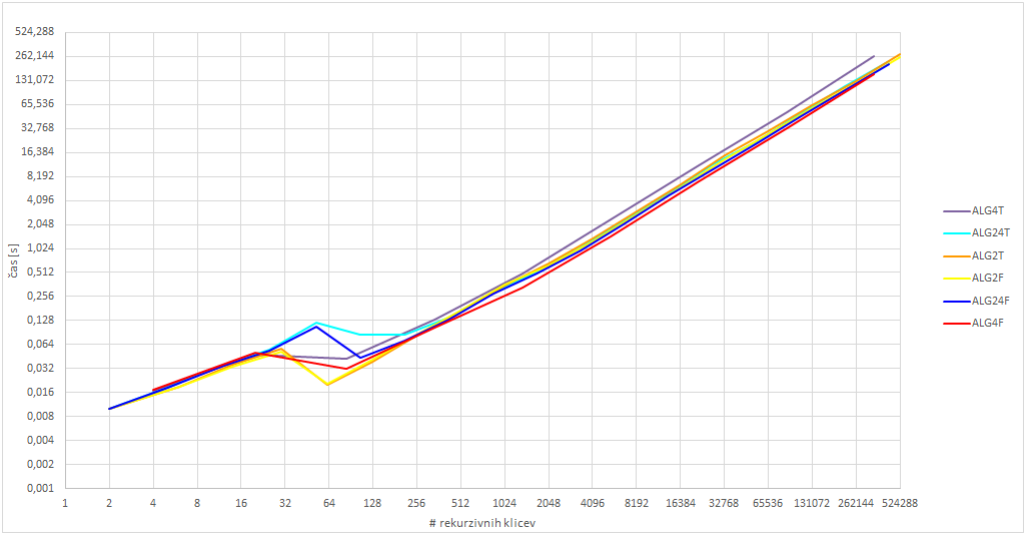
Pri algoritmih, ki smo jih implementirali na rekurzivni način, nas je zanimalo, kako število rekurzivnih klicev vpliva na čas izvajanja algoritma, kar je

prikazano na grafu 5.1. Vidimo, da je število klicev pri osnovi 2 največje, pri algoritmu z osnovo 4 pa najmanjše, kar je posledica načina delitve vhodnega problema na manjše podprobleme. Število rekurzivnih klicev je neodvisno od prostora decimacije (časovni ali frekvenčni).



Slika 5.1: Grafični prikaz števila rekurzivnih klicev v odvisnosti od dolžine vhodnih podatkov.

Spodnji graf 5.2 prikazuje srednjo vrednost časa izvajanja rekurzivnih algoritmov v odvisnosti od števila rekurzivnih klicev. Vidimo, da čas narašča premosorazmerno s številom klicev. Sklepamo lahko, da poleg števila klicev na čas izvajanja vpliva še neka druga lastnost, saj imajo algoritmi z enako osnovo različne čase izvajanja (čas je očitno odvisen tudi od prostora decimacije).



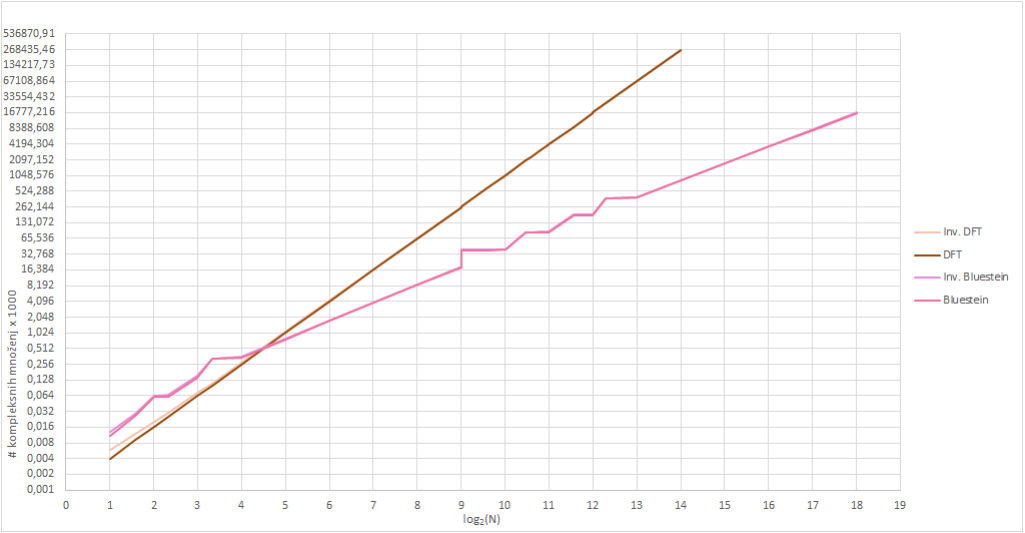
Slika 5.2: Grafični prikaz srednje vrednosti časa izvajanja algoritma v odvisnosti od števila rekurzivnih klicev.

5.2 Število kompleksnih operacij

Zanimalo nas je tudi število kompleksnih operacij v posamezni implementaciji algoritma. S pomočjo sistema ALGator smo enostavno prešteli število kompleksnih seštevanj (oz. odštevanj) in množenj (oz. deljenj).

5.2.1 Število kompleksnih množenj

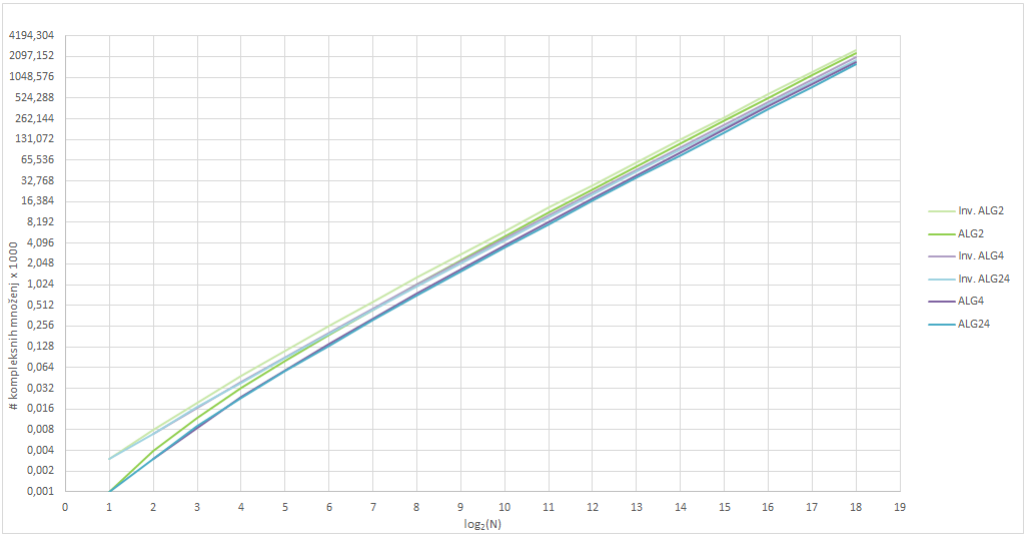
Opomnimo, da se za izračun transformacije in inverzne transformacija uporablja enaka funkcija (opisano v poglavju 3), edina razlika bo v številu kompleksnih množenj. To tudi vpliva na čas izvajanja. Iz tega je logično sklepati, da bo čas računanja inverzne transformacije vedno večji od časa računanja transformacije. Spodnji graf 5.3 jasno prikazuje, kako število kompleksnih množenj v DFT (implementirani po definiciji) eksponentno narašča.



Slika 5.3: Grafični prikaz števila kompleksnih množenj v odvisnosti od dolžine vhodnih podatokv za DFT in Bluesteinov algoritem.

Število kompleksnih množenj je odvisno od osnove (2, 4, 2 in 4) algoritma. Prav tako bi radi dodali, da je število kompleksnih množenj enako ne glede na prostor decimacije. Z grafa 5.4 je mogoče razbrati, da ima najmanjše število

množenj algoritem z deljeno osnovo (2 in 4), največje pa algoritem z osnovo 2. Vidimo, da je število kompleksnih množenj v inverznih transformacijah res večje za faktor N.



Slika 5.4: Grafični prikaz števila kompleksnih množenj v odvisnoti od dolžine vhodnih podatkov za vse implementirane algoritme in njihove inverze (razen DFT in Bluestainovega algoritma - slika 5.3)

5.2.2 Število kompleksnih seštevanj

Algoritmi implementirani po metodi Cooley-Tukey imajo enako število kompleksnih seštevanj. To lastnost smo opazili šele, ko smo si ogledali rezultate meritev. Z nekaj znanja matematike lahko enostavno pokažemo, da to res drži. Spomnimo se števila kompleksnih seštevanj posameznega algoritma iz poglavja 3. Vemo, da je število kompleksnih seštevanj neodvisno od prostora decimacije. Če z I označimo število kompleksnih seštevanj v algoritmu z osnovo 2 (3.50), z J v algoritmu z osnovo 4 (3.50) in s K v algoritmu z deljeno osnovo (3.60), potem je:

$$I = N \log_2 N, \tag{5.1}$$

$$J = 2N \log_4 N \tag{5.2}$$

in

$$K = 3\frac{N}{2} \log_4 N + \hat{I}. \tag{5.3}$$

Kjer je $\hat{I}$ dodatno število kompleksnih seštevanj v algoritmu z deljeno osnovo, ki jih pridelamo, ko uporabimo algoritem z osnovo 2 na najmanjših podproblemih (v zadnjem koraku rekurzije - velikosti 2), to je:

$$\hat{I} = \frac{N}{2} \log_4 N. \tag{5.4}$$

Dokaz. Spomnimo se formule za prehod na novo osnovo logaritma:

$$\log_a x = \frac{\log_b x}{\log_b a}, \tag{5.5}$$

kjer so $a \neq 1, b \neq 1$ in hkrati velja $a, b, x > 0$.
Sedaj v zgornji enačbi (5.1) nadomestimo algoritem z osnovo 2 z algoritmom z osnovo 4, potem dobimo:

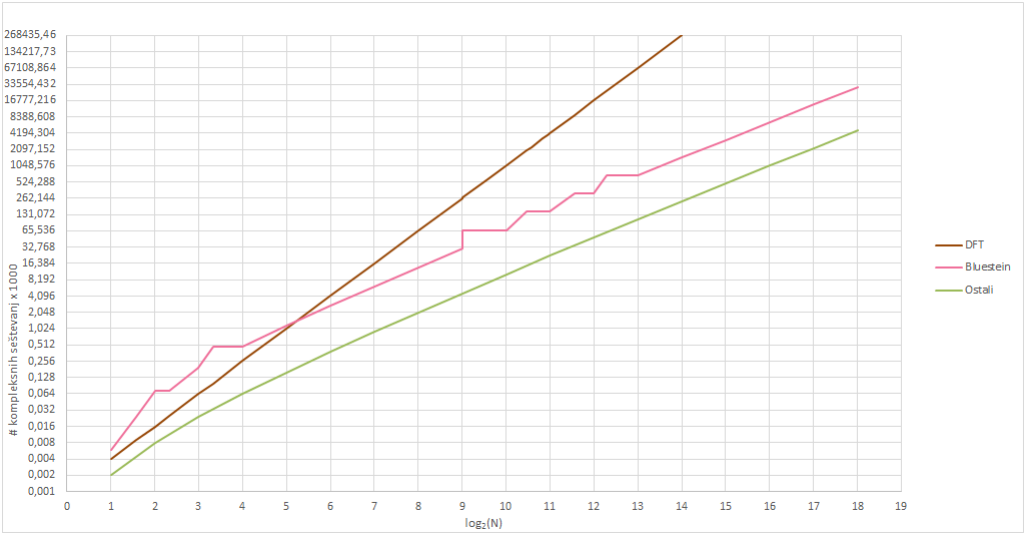
$$I = N \log_2 N = N\frac{\log_4 N}{\log_4 2} = 2N \log_4 N \implies I = J. \tag{5.6}$$

Takoj opazimo, da je število kompleksnih seštevanj v algoritmu z osnovo 2 enako kot v algoritmu z osnovo 4.
Poenostavimo enačbo števila kompleksnih seštevanj v algoritmu z deljeno osnovo:

$$\begin{aligned} K &= 3\frac{N}{2} \log_4 N + \hat{I} \\ &= 3\frac{N}{2} \log_4 N + \frac{N}{2} \log_4 N = (3\frac{N}{2} + \frac{N}{2}) \log_4 N \\ &= 2N \log_4 N \implies K = J = I. \end{aligned} \tag{5.7}$$

□

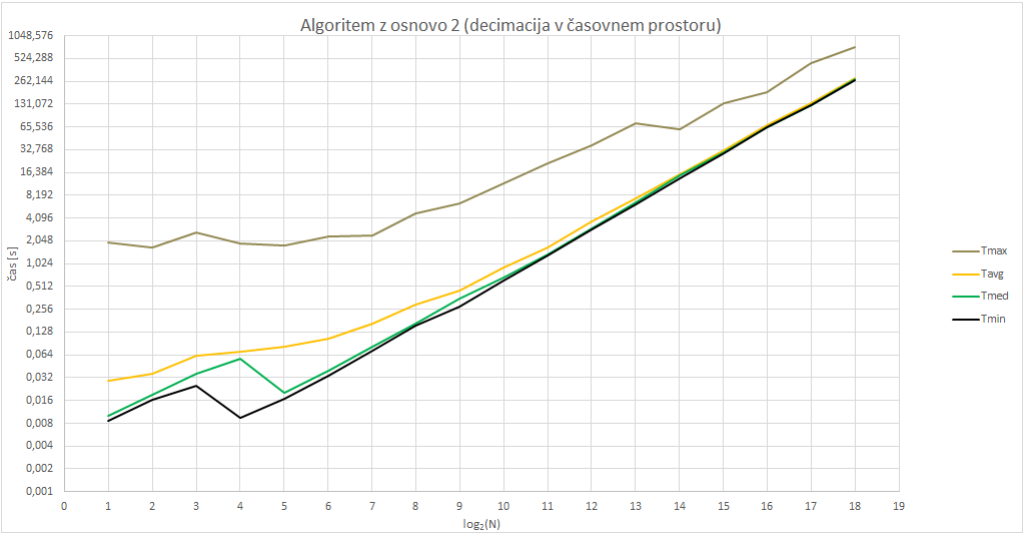
Spodnji graf 5.5 prikazuje, kako hitro narašča število kompleksnih seštevanj v navadni DFT in v Bluesteinovem algoritmu v primerjavi z algoritmi, ki uporabljajo metodo Cooley-Tukey.



Slika 5.5: Grafični prikaz števila kompleksnih seštevanj v odvisnosti od dolžine vhodnih podatkov.

5.3 Čas izvajanja

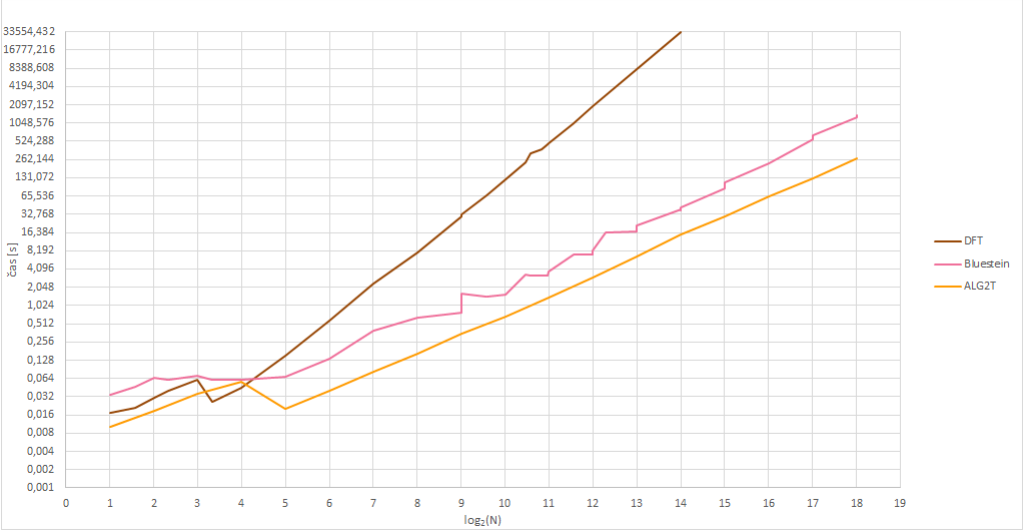
Ker vemo, da sistem ALGator na posameznem testnem primeru meri čas izvajanja T-krat (kolikokrat je podano v konfiguracijski datoteki testne množice) in da prihaja do razlik med prvim izvajanjem algoritma in nadaljnjimi, so nas zanimale tudi razlike med maksimalnim, minimalnim, povprečnim in srednjo vrednostjo časa izvajanja algoritma. Z grafa 5.6, ki prikazuje zgoraj navedene izmerjene čase izvajanja algoritma z osnovo 2 (decimacija v časovnem prostoru), je razvidno, da je maksimalni čas izvajanja za velike N malo manj kot trikrat večji od srednje vrednosti časa izvajanja algoritma. Zaradi vrednosti, ki so prikazane na spodnjem grafu, smo se odločili, da se bomo pri meritvah časa osredotočili na srednjo vrednost izmerjenega časa. To pomeni, da so vsi prikazani časi na grafih v tej nalogi srednje vrednosti izmerjenih časov.



Slika 5.6: Primerjava minimalnega, maksimalnega, povprečnega in srednje vrednosti časa izvajanja algoritma z osnovo 2 z decimacijo v časovnem prostoru (rekurzivna implementacija) v odvisnosti od dolžine vhodnih podatkov.

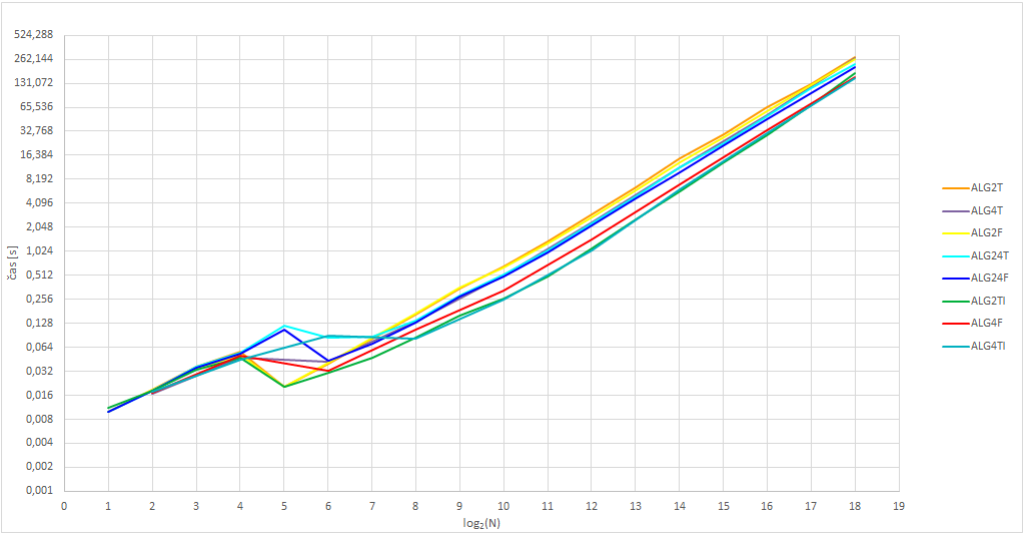
Rekurzivni algoritem z osnovo 2 z decimacijo v časovnem prostoru je najpočasnejši

med algoritmi, ki smo jih implementirali po metodi Cooley-Tukey. Vendar je vseeno veliko hitrejši kot navadna DFT ali Bluesteinov algoritem, kar jasno pokaže graf 5.7.



Slika 5.7: Srednja vrednost časa izvajanja DFT, Bluesteinovega algoritma in algoritma z osnovo 2 z decimacijo v časovnem prostoru.

Zanimiva je ugotovitev z grafa 5.8, da so algoritmi (implementirani na rekurzivni način), ki uporabljajo decimacijo v frekvenčnem prostoru, hitrejši od tistih z decimacijo v časovnem prostoru, kljub enakemu številu operacij. Prav tako tudi, da je algoritem z osnovo 4 hitrejši od algoritma z deljeno osnovo, saj za izračun potrebuje večje število kompleksnih množenj. Do $N < 2^7$ je celo hitrejši od iterativne implementacije algoritma z osnovo 4 (z decimacijo v časovnem prostoru).



Slika 5.8: Grafični prikaz srednje vrednosti časa izvajanja v odvisnosti od dolžine vhodnih podatkov algoritmov, ki uporabljajo metodo Cooley-Tukey.

5.3.1 Napoved časa izvajanja rekurzivnih algoritmov glede na število rekurzivnih klicev

Graf 5.2 iz prejšnjega razdelka 5.1 prikazuje (približno) logaritmično (Alog(A)) odvisnost časa izvajanja algoritma od števila rekurzivnih klicev. Zaradi te lepe lastnosti in ker je čas izvajanja algoritma težko (pošteno) izmeriti, smo se odločili, da ga bomo poskušali napovedati s pomočjo števila rekurzivnih klicev. Za napoved smo najprej potrebovali računsko oceniti število klicev v algoritmih. Z opazovanjem izmerjenega števila klicev in poznavanjem geometrijskih vrst smo prišli do naslednjih formul:

- **algoritma z osnovo 2**:

$$\#CALL_2(N) = 2\frac{1-N}{1-2} \tag{5.8}$$

- **algoritma z osnovo 4**:

$$\#CALL_4(N) = 4\frac{1-N}{1-4} \tag{5.9}$$

- **algoritma z deljeno osnovo**:

$$\#CALL_{24}(N) = \begin{cases} 2 & , \text{če } N = 2 \\ 5 & , \text{če } N = 4 \\ 2 \cdot \#CALL_{24}(\frac{N}{4}) + \#CALL_{24}(\frac{N}{2}) + 3 & , \text{če } N > 5 \end{cases} \tag{5.10}$$

kjer je N ustrezna dolžina vhodnih podatkov, ki jih sprejme algoritem. Iz tabele na sliki 5.9 vidimo, da se število izračunanih rekurzivnih klicev popolnoma ujema z izmerjenim številom.

	ALG2T		ALG2F		ALG4T		ALG4F		ALG24T		ALG24F	
N	*Izmerjen #CALL*	*Izračunan #CALL*	*Izmerjen #CALL*	*Izračunan #CALL*	*Izmerjen #CALL*	*Izračunan #CALL*	*Izmerjen #CALL*	*Izračunan #CALL*	*Izmerjen #CALL*	*Izračunan #CALL*	*Izmerjen #CALL*	*Izračunan #CALL*
2	2	2	2	2	/	/	/	/	2	2	2	2
4	6	6	6	6	4	4	4	4	5	5	5	5
8	14	14	14	14	/	/	/	/	12	12	12	12
16	30	30	30	30	20	20	20	20	25	25	25	25
32	62	62	62	62	/	/	/	/	52	52	52	52
64	126	126	126	126	84	84	84	84	105	105	105	105
128	254	254	254	254	/	/	/	/	212	212	212	212
256	510	510	510	510	340	340	340	340	425	425	425	425
512	1022	1022	1022	1022	/	/	/	/	852	852	852	852
1024	2046	2046	2046	2046	1364	1364	1364	1364	1705	1705	1705	1705
2048	4094	4094	4094	4094	/	/	/	/	3412	3412	3412	3412
4096	8190	8190	8190	8190	5460	5460	5460	5460	6825	6825	6825	6825
8192	16382	16382	16382	16382	/	/	/	/	13652	13652	13652	13652
16384	32766	32766	32766	32766	21844	21844	21844	21844	27305	27305	27305	27305
32768	65534	65534	65534	65534	/	/	/	/	54612	54612	54612	54612
65536	131070	131070	131070	131070	87380	87380	87380	87380	109225	109225	109225	109225
131072	262142	262142	262142	262142	/	/	/	/	218452	218452	218452	218452
262144	524286	524286	524286	524286	349524	349524	349524	349524	436905	436905	436905	436905

Slika 5.9: Število izmerjenih in izračunanih rekurzivnih klicev za posamezen algoritem.

Z metodo najmanjših kvadratov smo potem iz podatkov z grafa 5.2 za vsak algoritem izračunali konstanti a in b:

$$b = \frac{r\sum_{i=1}^{r}(y_i \ln(x_i)) - \sum_{i=1}^{r} y_i \sum_{i=1}^{r} \ln(x_i)}{r\sum_{i=1}^{r} \ln(x_i)^2 - (\sum_{i=1}^{r} \ln(x_i))^2}, \tag{5.11}$$

$$a = \frac{\sum_{i=1}^{r} y_i - b\sum_{i=1}^{r} \ln(x_i)}{r}. \tag{5.12}$$

Kjer je r število prvih r točk uporabljenih za izračun konstant a in b, ki je odvisno od osnove algoritma:

- osnova 2: r = 9,
- osnova 4: r = 6,
- deljena osnova: r = 8.

Dejanske izračunane vrednosti konstant za posamezen algoritem so prikazane v tabeli na sliki 5.10.

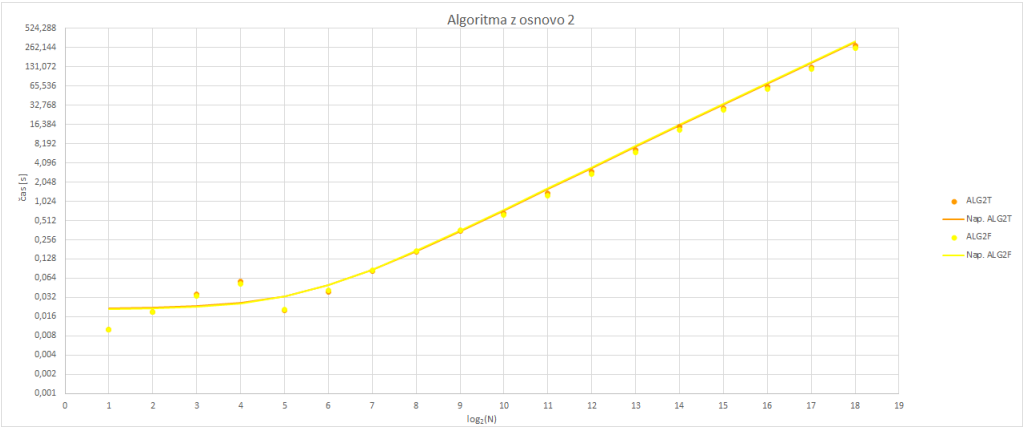
	ALG2		ALG4		ALG24	
	DIT	*DIF*	*DIT*	*DIF*	*DIT*	*DIF*
a	21,71	20,93157673	28,35827028	32,2245231	46,86036066	38,47599888
b	0,046184767	0,04741944	0,04914152	0,030322837	0,038755649	0,035481334

Slika 5.10: Izračunane vrednosti konstant a in b.

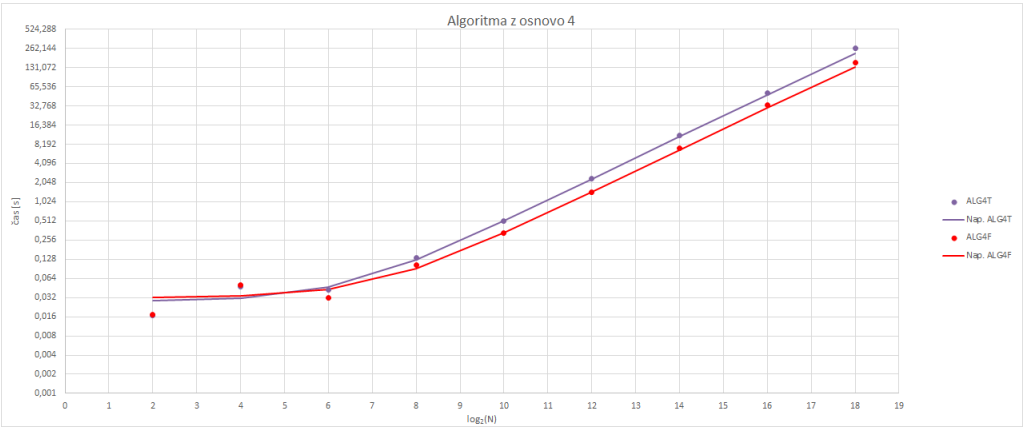
Nato smo z logaritmičnim ujemanjem po spodnji formuli izračunali napovedan čas izvajanja za vsak algoritem:

$$T(N) = a + b \cdot \#CALL_q(N) \cdot \ln(\#CALL_q(N)). \tag{5.13}$$

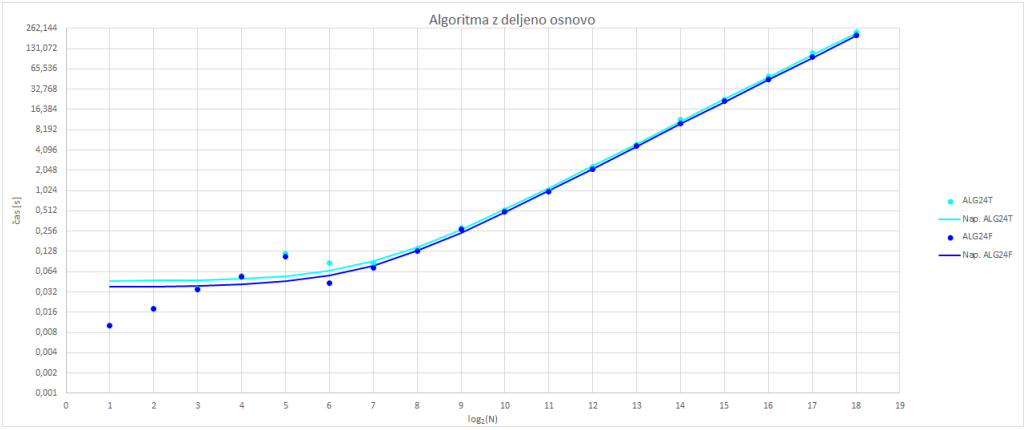
S spodnjih grafov 5.11, 5.12, 5.13 razberemo, da se napovedan čas dobro ujema z izmerjenim.



Slika 5.11: Grafični prikaz med izmerjenim in napovedanim (izračunanim) časom izavanja za algoritma z osnovo 2.

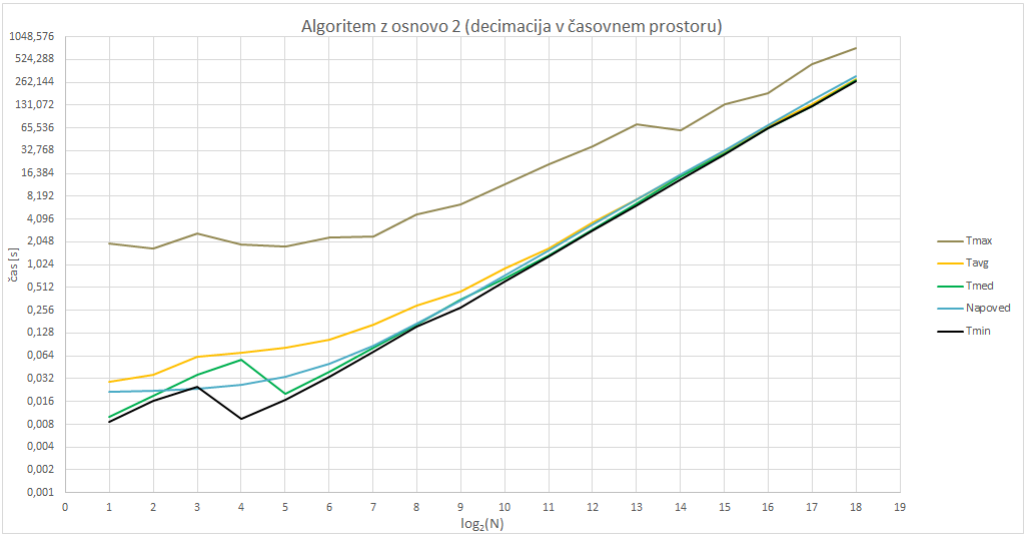


Slika 5.12: Grafični prikaz med izmerjenim in napovedanim (izračunanim) časom izavanja za algoritma z osnovo 4.



Slika 5.13: Grafični prikaz med izmerjenim in napovedanim (izračunanim) časom izavanja za algoritma z deljeno osnovo.

Za primerjavo z ostalimi izmerjenimi časi algoritma z osnovo 2 DIT smo vključili še graf 5.14. Razlika pri ostalih algoritmih je prav tako zelo podobna.



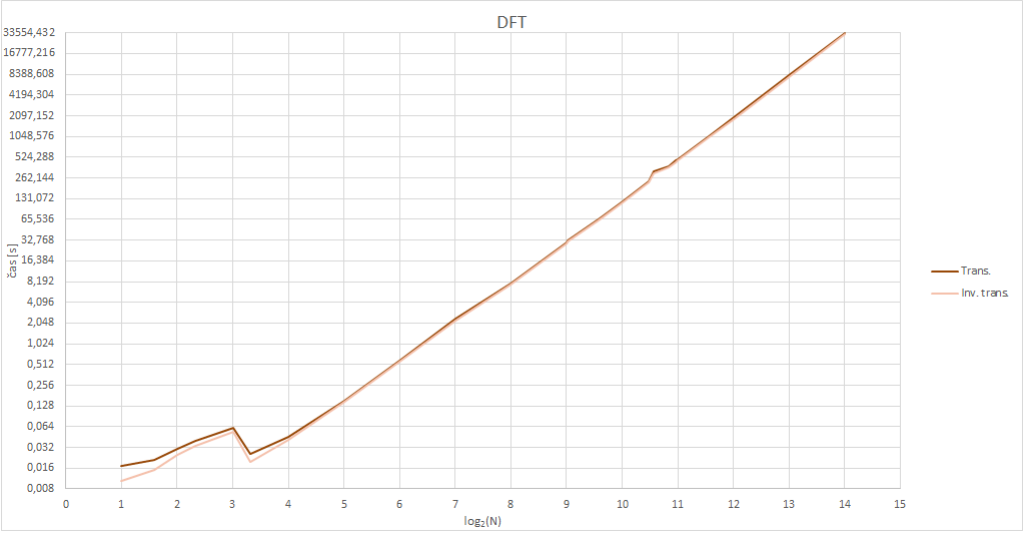
Slika 5.14: Grafični prikaz izmerjenih časov in napovedanega (izračunanega) za algoritem z osnovo 2 (DIT).

5.3.2 Primerjava časa izvajanja transformacije in inverzne transformacije

Kljub večjemu številu kompleksnih množenj, ki so potrebna za izračun inverza, se je pri nekaterih dolžinah vhodnih podatkov (N) izračun inverzne transformacije vseeno izvedel hitreje kot izračun transformacije. Opazili smo tudi večjo razliko v času izvajanja med inverzno transformacijo in transformacijo pri algoritmih, ki uporabljajo decimacijo v frekvenčnem prostoru.

DFT

Najbolj nas je presenetil čas izavajanja DFT (po definiciji) in njegovega inverza. Rezultati na grafu 5.15 kažejo, da se izračun inverza izvede hitreje ne glede na dolžino vhodnih podatkov. To ni najbolj logično, saj za izračun inverza potrebujemo več kompleksnih množenj.

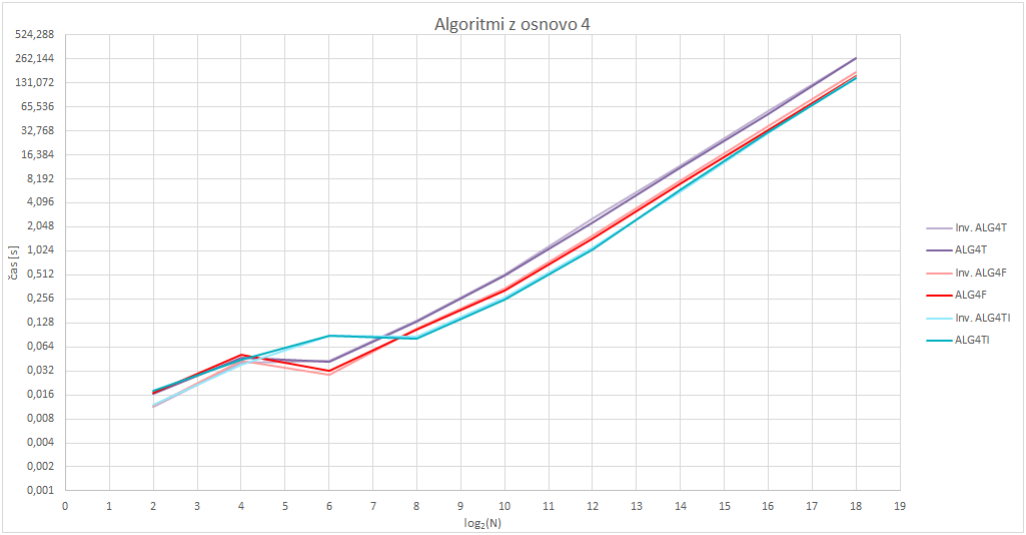


Slika 5.15: Grafični prikaz srednje vrednosti časa izvajanja DFT in inverza.

Algoritmi z osnovo 4

Na grafu 5.16 smo opazili podobne lastnosti tudi pri algoritmih z osnovo 4, da se inverz do določene velikosti vhodnih podatkov računa hitreje in sicer:

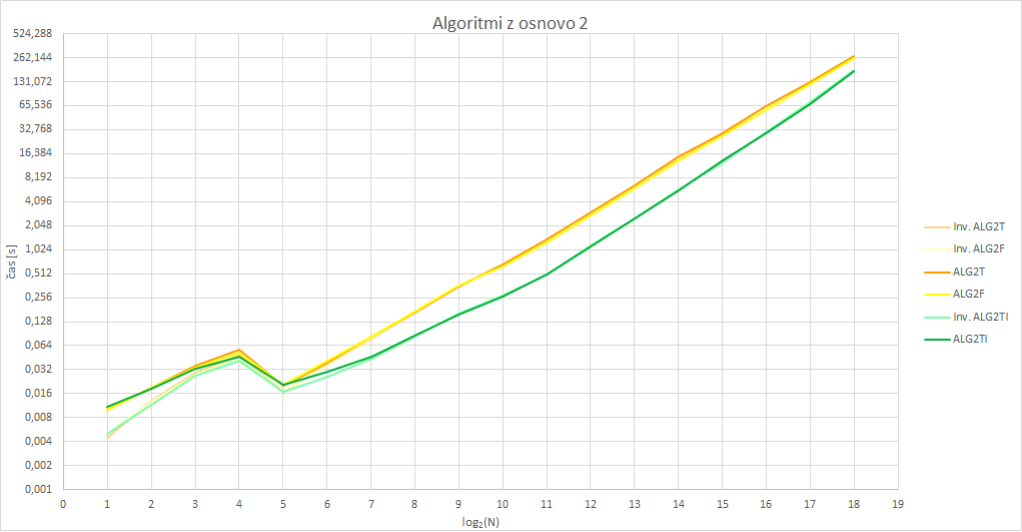
- za algoritem z decimacijo v časovnem prostoru velja, da je izvajanje inverza hitrejše do dolžine 2^6 (tudi pri iterativni različici),
- za algoritem z decimacijo v frekvenčnem prostoru pa je izračun inverza hitrejši do dolžine 2^8.



Slika 5.16: Grafični prikaz srednje vrednosti časa izvajanja v odvisnosti od dolžine vhodnih podatkov algoritmov z osnovo 4 in njihovih inverznih transformacij.

Algoritmi z osnovo 2

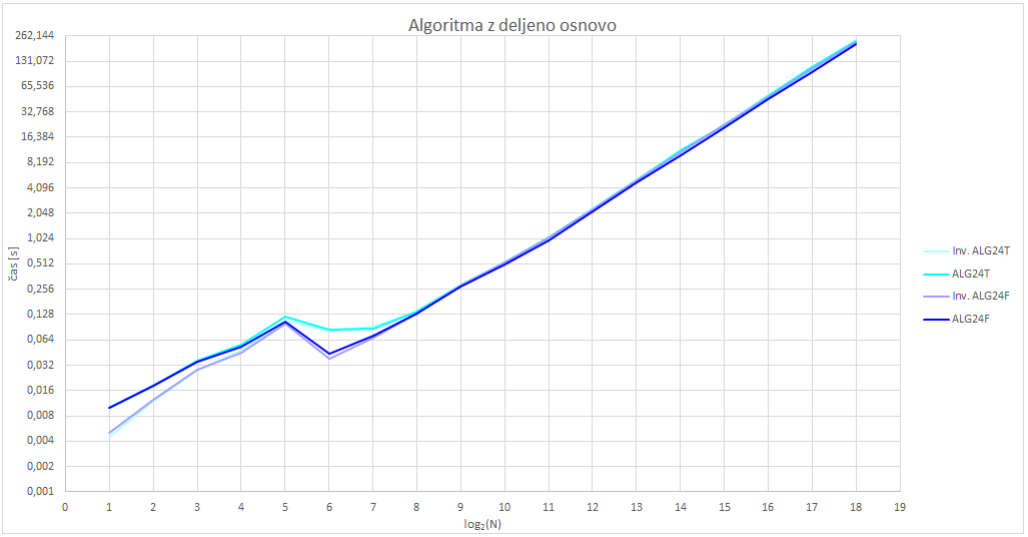
Za algoritme z osnovo 2 neodvisno od prostora decimacije in načina implementacije (iterativno/rekurzivno) se je izkazalo, da je izračun inverza hitrejši pri podatkih z velikostjo (N), ki je manjša ali kvečjemu enaka od 2^8. To je vidno na grafu 5.17.



Slika 5.17: Grafični prikaz srednje vrednosti časa izvajanja v odvisnosti od dolžine vhodnih podatkov algoritmov z osnovo 2 in njihovih inverznih transformacij.

Algoritma z deljeno osnovo

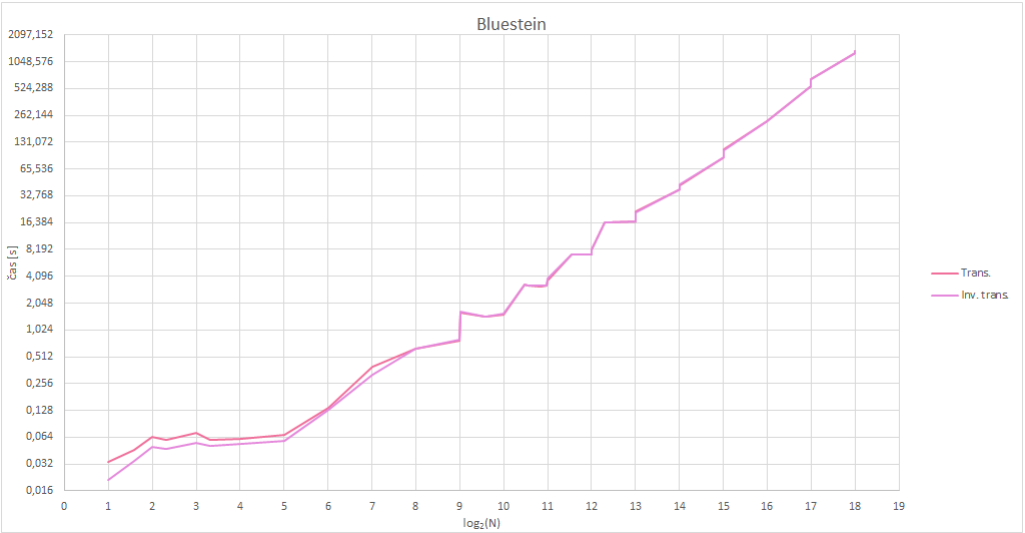
Z grafa 5.18 razberemo, da je pri algoritmih z deljeno osnovo (neodvisno od prostora decimacije) izračun inverzne transformacije hitrejši, dokler je dolžina vhodnih podatkov manjša od 2^{10}.



Slika 5.18: Grafični prikaz srednje vrednosti časa izvajanja v odvisnosti od dolžine vhodnih podatkov algoritmov z deljeno osnovo in njunih inverznih transformacij.

Bluesteinov algoritem

Pri Bluesteinovem algoritmu za vhodne podatke velikosti $N > 2^8$ prihaja do minimalnih razlik v času izvajanja. Preden smo začeli z raziskovanjem smo takšen rezultat pričakovali pri vseh algoritmih, ne glede na velikost vhodnih podatkov (N), vendar rezultati meritev kažejo drugače. Ker ta algoritem izračuna transformacijo s pomočjo konvolucije, za kar uporablja algoritem z osnovo 2 (DIT - iterativno različico), opazimo tudi na grafu 5.19 enako lastnost, kot na zgornjem grafu 5.17 pri algoritmih z osnovo 2, da je izračun inverza hitrejši pri $N < 2^8$.



Slika 5.19: Grafični prikaz srednje vrednosti časa izvajanja (transformacije in inverzne transformacije) v odvisnosti od dolžine vhodnih podatkov Bluesteinovega algoritma.

Poglavje 6

Sklepne ugotovitve

S pomočjo sistema ALGator smo na vhodnih podatkih dolžine od 2^2 pa do 2^{18} testirali 10 različnih algoritmov za izračun Fourierjeve transformacije in njenega inverza. Po pričakovanjih se je najslabše odrezal algoritem, ki smo ga implementirali striktno po definiciji DFT. Med algoritmi, ki smo jih implementirali z rekurzijo, se je najbolje odrezal algoritem z osnovo 4 z decimacijo v frekvenčnem prostoru, kljub temu, da algoritem z deljeno osnovo potrebuje manj množenj. Izmed iterativnih različic je bil prav tako najhitrejši algoritem z osnovo 4 (z decimacijo v časovnem prostoru). Ugotovili smo, da se izmed implementiranih algoritmov za dolžino vhodnih podatkov vključno in manjšo od 2^{17} najbolj splača uporabiti iterativno različico algoritma z osnovo 2 (z decimacijo v časovnem prostoru), za podatke daljše od 2^{17} pa algoritem z osnovo 4.

Rezultati pri izračunu časa izvajanja v odvisnosti od števila rekurzivnih klicev algoritmov kažejo zelo dobro ujemanje napovedanih (izračunanih) in izmerjenih vrednosti. Imajo enak trend kot izmerjen povprečni čas izavanja. Čas izvajanaj smo merili v 18 točkah, krivuljo za napoved pa smo izračunali iz manj kot prve polovice izmerjenih točk, kar kaže na dobro metodo napovedi.

Pri primerjavah časa izvajanja transformacij in inverznih transformacij smo opazili, da so razlike v času izvajanja med inverzom in transformacijo večje pri algoritmih, ki uporabljajo decimacijo v frekvenčnem prostoru, kot pa pri

algoritmih, ki uporabljajo decimacijo v časovnem prostoru.
Glede na zgornje ugotovitve bi bilo zanimivo analizirati, kako bi se obnesel algoritem z osnovo 4 z decimacijo v frekvenčnem prostoru implementiran na iterativen način. Zanima nas tudi, kakšni bi bili rezultati, če bi implementirati enake algoritme v programskem jeziku C. S posebnim delom ALGatorja, ki se imenuje ALGatorc in omogoča izvajanje ter analiziranje algoritmov, ki so implementirani v programsekm jeziku C, bi primerjavo naredili na enostaven način.
Kljub dobri zasnovi sistema ALGator pa smo ob uporabi opazili še nekaj pomanjkljivosti, ki bi jih bilo potrebno odpraviti. Eden izmed večjih problemov je uporaba grafičnega prikaza rezultatov testiranja, saj se že ob manjši izbiri podatkov grafični vmesnik odziva zelo počasi. Prav tako je oteženo približevanje na grafu. Pogrešali smo tudi numerične vrednosti na obeh oseh, ko smo približali določen del grafa.
Ta hiba sistema nas je tudi prislila, da smo za generiranje slik grafov uporabili drugo orodje.

Literatura

[1] M. Beck et al., "A First Course in Complex Analysis". [Online]. Dosegljivo:
http://math.sfsu.edu/beck/papers/complex.pdf. [Dostopano 27. 2. 2016].

[2] Michael M. Dougherty, J. Gieringer "First Year Calculus For Students of Mathematics and Related Disciplines", str. 747–780. [Online]. Dosegljivo:http://faculty.swosu.edu/michael.dougherty/book/book.pdf. [Dostopano 5. 3. 2016].

[3] D. Cherney, T. Denton in A. Waldron "Linear Algebra", 1. izd., str. 143, Davis California, 2013.

[4] Robert M. Gray, "Toeplitz and Circulant Matrices: A review", *Foundations and Trends in Communications and Information Theory*, vol. 2, str. 1–34, Stanford: Stanford University, 2006.

[5] T. Dobravec, "ALGator - izvajanje in analiza algoritmov". [Online]. Dosegljivo:
https://github.com/ALGatorDevel/Algator. [Dostopano 18. 2. 2016].

[6] Apache Commons. Commons Math: The Apache Commons Mathematics Library. [Online]. Dosegljivo:
http://commons.apache.org/proper/commons-math/. [Dostopano 19. 2. 2016].

[7] Henri J. Nussbaumer, "Fast Fourier Transform and Convolution Algorithms: Second Corrected and Updated Edition", 2. izd., *Springer Series in Information Science*, vol. 2, str. 81–94, Berlin [etc.]: Springer, 1982.

[8] E. Chu, A. George "Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms", *Computational Mathematics Series*, Boca Raton: CRC Press LLC, 2000.

[9] John G. Proakis, Dimitris K. Manolakis "Digital Signal Processing: Principles, Algorithms, and Applications", 4. izd., New Jersey: Prentice Hall, 7(4), 2006.

[10] P. Duhamel, Henk D. L. Hollmann. "Split radix FFT algorithm". [Online]. Dosegljivo: https://www.researchgate.net/publication/3399594_Split_radix´_FFT_algorithm. [Dostopano 22. 2. 2016].

[11] Ž. Zorman "Izvorna koda implementiranih algoritmov". [Online]. Dosegljivo: https://github.com/Flinch010/bachelor-s-degree. [Dostopano 4. 2. 2017].